✓ **Program 1 – Program to traverse the elements of an array**

```
#include<stdio.h>
#include<conio.h>
int main()
{
        int i, n, arr[20];
        clrscr();
        printf("\n Enter the number of elements in the array:");
        scanf("%d",&n);
        printf("\n Enter the elements in the array:");
        for(i=0;i<n;i++)
        {
                printf("\n arr[%d] =",i);
                scanf("%d",&arr[i]);
        }
        printf("\n The array elements are:");
        for(i=0;i<n;i++)
        {
                printf("\t %d",arr[i]);
        }
        getch();
        return 0;
}
```

✓ **Program 2 - Program of search operation to search for an element in an array**

```
#include<stdio.h>
#include<conio.h>
int main()
{
   int n, item, i, j=0, arr[40];
   clrscr();
   printf("\n Enter the number of elements in array: ");
   scanf("%d",&n);
   printf("\n Enter the elements in array: ");
   for(i=0;i<n;i++)
   {
        printf("\n arr[%d] = ",i);
        scanf("%d",&arr[i]);
   }
   printf("\n Element to be searched = ");
   scanf("%d",&item);
   while(j<n)
   {
        if(arr[j]==item)
        {
```

```
                printf("\n Element %d found at index %d\n",item,j);
                break;
        }
        j++;
  }
  if(j==n)
  {
        printf("\n Element %d not found in the array",item);
  }
  getch();
  return 0;
}
```

✓ **Program 3 – Program to insert an element in an array**

```
#include<stdio.h>
#include<conio.h>
int main()
{
        int i, n, num, pos, arr[10];
        clrscr();
        printf("\n Enter the number of elements in array:");
        scanf("%d",&n);
        printf("\n Enter the elements in array: ");
        for(i=0;i<n;i++)
        {
                printf("\n arr[%d]= ",i);
                scanf("%d",&arr[i]);
        }
        printf("\n Enter the number to be inserted:");
        scanf("%d", &num);
        printf("\n Enter the position at which the number has to be inserted:");
        scanf("%d",&pos);
        for(i=n-1;i>=pos;i--)
        {
                arr[i+1]=arr[i];
        }
        arr[pos] = num;
        n = n+1;
        printf("\n The array after insertion of %d is: ",num);
        for(i=0;i<n;i++)
        {
                printf("\n arr[%d] = %d", i, arr[i]);
        }
        getch();
        return 0;
}
```

✓ **Program 4 – Program to delete an element in an array**

```
#include<stdio.h>
#include<conio.h>
int main()
{
	int i, n, pos, arr[100];
	clrscr();
	printf("\n Enter the number of elements in array: ");
	scanf("%d",&n);
	printf("\n Enter the elements in array: ");
	for(i=0;i<n;i++)
	{
		printf("\n arr[%d] = ",i);
		scanf("%d",&arr[i]);
	}
	printf("\n Enter the position at which the element will be deleted:");
	scanf("%d",&pos);
	for(i=pos;i<n-1;i++)
	{
		arr[i] = arr[i+1];
		n--;
	}
	printf("\n The array after deletion is: ");
	for(i=0;i<n;i++)
	{
		printf("\n arr[%d] = %d", i, arr[i]);
	}
	getch();
	return 0;
}
```

✓ **Program 5 - Program for linear searching**

```
#include<stdio.h>
#include<conio.h>
int main()
{
	int arr[100], n, i, target, found=0;
	clrscr();
	printf("\n Enter the number of elements in an array:");
	scanf("%d",&n);
	printf("\n Enter the elements in array: ");
	for(i=0;i<n;i++)
	{
		printf("\n arr[%d] = ",i);
		scanf("%d",&arr[i]);
```

```
	}
	printf("Enter the value to be searched: ");
	scanf("%d",&target);
	//linear search algorithm
	for(i=0;i<n;i++)
	{
		if(arr[i]==target)
		{
			printf("Element %d found at index %d\n",target,i);
			found = 1;
			break;
		}
	}
	if(!found)
	{
		printf("Element %d not found in the array.\n",target);
	}
	getch();
	return 0;
}
```

✓ **Program 6 - Program for binary search**

```
#include<stdio.h>
#include<conio.h>
void main()
{
	int arr[100], n, i, target;
	clrscr();
	printf("\n Enter the number of elements in an array:");
	scanf("%d",&n);
	printf("\n Enter the elements in array: ");
	for(i=0;i<n;i++)
	{
		printf("\n arr[%d] = ",i);
		scanf("%d",&arr[i]);
	}
	//Sorting array using bubble sort (extra code to sort if element are not sorted)
	for(i=0;i<n-1;i++)
	{
		for(int j=0;j<n-i-1;j++)
		{
			if(arr[j]>arr[j+1])
			{
				//swap
				int temp=arr[j];
				arr[j]=arr[j+1];
				arr[j+1]=temp;
```

```
                        }
                }
        }
        printf("Enter the value to search:");
        scanf("%d",&target);
        //binary search algorithm
        int beg=0, end=n-1, mid, found=0;
        while(beg <= end)
        {
                mid = beg + (end-beg)/2;
                if(arr[mid]==target)
                {
                        printf("Element %d found at index %d\n",target,mid);
                        found = 1;
                        break;
                }
                else if(arr[mid] < target)
                {
                        beg = mid + 1;
                }
                else
                {
                        end = mid - 1;
                }
        }
        if(!found)
        {
                printf("Element %d not found in the array:\n",target);
        }
        getch();
}
```

- ✓ **Program 7 - Program for selection sort**

```
#include<stdio.h>
#include<conio.h>
void main()
{
        int arr[100], n, i, j, minIndex, temp;
        clrscr();
        printf("\n Enter the number of elements in an array:");
        scanf("%d",&n);
        printf("\n Enter the elements in array: ");
        for(i=0;i<n;i++)
        {
                printf("\n arr[%d] = ",i);
                scanf("%d",&arr[i]);
        }
```

```
	//Selection sort algorithm
	for(i=0;i<n-1;i++)
	{
		minIndex = i;
		for(j=i+1;j<n;j++)
		{
			if(arr[j]<arr[minIndex])
			{
				minIndex=j;
			}
		}
		if(minIndex != i)
		{
			temp = arr[i];
			arr[i] = arr[minIndex];
			arr[minIndex] = temp;
		}
	}
	printf("Sorted Array after Selection Sort: \n");
	for(i=0;i<n;i++)
	{
		printf("%d\n",arr[i]);
	}
	printf("\n");
	getch();
}
```

Selection sort program

```
#include <stdio.h>
#include <conio.h>
void main()
{
  clrscr();
  int array[100], n, i, j, position, t;
  printf("Enter number of elements\n");
  scanf("%d", &n);

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

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

  for (i = 0; i < (n - 1); i++) // finding minimum element
(n-1) times
  {
    position = i;

    for (j = i + 1; j < n; j++)
    {
      if (array[position] > array[j])
        position = j;
    }
    if (position != i)
    {
      t = array[i];
      array[i] = array[position];
      array[position] = t;
    }
  }

  printf("Sorted list in ascending order:\n");

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

  getch();
}
```

✓ **Program 8 - Program for Bubble Sort**

```
#include<stdio.h>
#include<conio.h>
void main()
{
        int arr[100], n, i, temp;
        clrscr();
        printf("\n Enter the number of elements in an array:");
        scanf("%d",&n);
        printf("\n Enter the elements in array: ");
        for(i=0;i<n;i++)
```

```
	{
		printf("\n arr[%d] = ",i);
		scanf("%d",&arr[i]);
	}
	//Sorting array using bubble sort
	for(i=0;i<n-1;i++)
	{
		for(int j=0;j<n-i-1;j++)
		{
			if(arr[j]>arr[j+1])
			{
				//swap
				int temp=arr[j];
				arr[j]=arr[j+1];
				arr[j+1]=temp;
			}
		}
	}
	printf("Sorted array after Bubble Sort: \n");
	for(i=0;i<n;i++)
	{
		printf("%d\n",arr[i]);
	}
	printf("\n");
	getch();
}
```

Bubble sort program

```c
/* Bubble sort code */
#include <stdio.h>

#include<conio.h>

void main()
{

  clrscr();
  int array[100], n, i, j, swap;

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

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

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

  for (i = 0 ; i < n - 1; i++)
  {
    for (j = 0 ; j < n - i - 1; j++)
    {
      if (array[j] > array[j+1]) /* For decreasing order use
'<' instead of '>' */
      {
        swap       = array[j];
        array[j]   = array[j+1];
        array[j+1] = swap;
      }
    }
  }

  printf("Sorted list in ascending order:\n");

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

  getch();
}
```

```
function mergeSort(array)
  if length of array <= 1
    return array
  middle = length of array / 2
  leftArray = mergeSort(first half of array)
  rightArray = mergeSort(second half of array)
  return merge(leftArray, rightArray)
```

✓ **Program 9 - Program for insertion sort**

#include<stdio.h>

```
#include<conio.h>
void main()
{
	int arr[100], n, i, j, key;
	clrscr();
	printf("\n Enter the number of elements in an array:");
	scanf("%d",&n);
	printf("\n Enter the elements in array: ");
	for(i=0;i<n;i++)
	{
		printf("\n arr[%d] = ",i);
		scanf("%d",&arr[i]);
	}
	//Insertion Sort Algorithm
	for(i=1;i<n;i++)
	{
		key=arr[i];
		j=i-1;
		while(j>=0&&arr[j]>key)
		{
			arr[j+1]=arr[j];
			j--;
		}
		arr[j+1]=key;
	}
	printf("Sorted Array after using Insertion Sort: \n");
	for(i=0;i<n;i++)
	{
		printf("%d\n",arr[i]);
	}
	printf("\n");
	getch();
}
```

- ✓ **Program 10 – Program for Merge Sort**

```
#include<stdio.h>
#include<stdlib.h>
#define MAX_SIZE 100
#include<conio.h>
void Merge(int arr[],int left,int mid,int right)
{
	int i,j,k;
	int size1=mid-left+1;
	int size2=right-mid;
	int Left[MAX_SIZE],Right[MAX_SIZE];
	for(i=0;i<size1;i++) Left[i]=arr[left+i]; for(j=0;j<size2;j++) Right[j]=arr[mid+1+j];
```

```
	i=0; j=0; k=left;
	while(i<size1 && j<size2)
	{
		if(Left[i]<=Right[j])
		{
			arr[k]=Left[i]; i++;
		}
		else
		{
			arr[k]=Right[j]; j++;
		}
		k++;
	}
	while(i<size1)
	{
		arr[k]=Left[i]; i++;
		k++;
	}
	while(j<size2)
	{
		arr[k]=Right[j]; j++; k++;
	}
}
void Merge_Sort(int arr[],int left,int right)
{
	if(left<right)
	{
		int mid=left+(right-left)/2; Merge_Sort(arr,left,mid);
		Merge_Sort(arr,mid+1,right);
		Merge(arr,left,mid,right);
	}
}
int main()
{
	clrscr();
	int arr[MAX_SIZE];
	int size,i;
	printf("Enter the size of array: ",MAX_SIZE);
	scanf("%d",&size);
	if(size>MAX_SIZE)
	{
		printf("Size exceeds the maximum limit of %d \n",MAX_SIZE);
		return 1;
	}
	printf("Enter the element of array:\n");
	for(i=0;i<size;i++)
	{
		scanf("%d",&arr[i]);
```

```
        }
        Merge_Sort(arr,0,size-1);
        printf("The sorted array is:\n");
        for(i=0;i<size;i++)
        {
                printf(" %d",arr[i]);
        }
        printf("\n");
        getch();
        return 0;
}
```

MERGE SORT

```
#include <stdio.h>


// Function to merge two sorted subarrays
void merge(int arr[], int l, int m, int r) {
    // Calculate sizes of two subarrays to be merged
    int n1 = m - l + 1;
    int n2 = r - m;

    // Create temporary arrays to hold the subarrays
    int L[n1], R[n2];

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

    // Merge the temporary arrays back into arr[l..r]
    int i = 0, j = 0, k = l;
    while (i < n1 && j < n2) {
        if (L[i] <= R[j]) {
```

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

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

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

// Recursive function to perform merge sort
void mergeSort(int arr[], int l, int r) {
    // Base case: If there is only one element, do nothing
    if (l < r) {
        // Find the middle point to divide the array into two halves
        int m = l + (r - l) / 2;

        // Recursively sort the left and right halves
```

```
        mergeSort(arr, l, m);
        mergeSort(arr, m + 1, r);

        // Merge the sorted halves
        merge(arr, l, m, r);
    }
}

int main() {
    int arr[] = {5, 10, 2, 9};
    int n = sizeof(arr) / sizeof(arr[0]);

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

    mergeSort(arr, 0, n - 1);

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

    return 0;
}
```

- ✓ **Program 11: Program for Quick Sort**

```
#include<stdio.h>
#include<conio.h>
void quicksort(int number[],int first,int last)
{
        int i, j, pivot, temp;
        if(first<last)
        {
                pivot=first;
                i=first;
                j=last;

                while(i<j)
                {
                        while(number[i] <= number[pivot] && i<last)
                                i++;
                        while(number[j] > number[pivot])
                                j--;
```

```
                        if(i < j){
                                temp=number[i];
                                number[i]=number[j];
                                number[j]=temp;
                        }
                }

                temp = number[pivot];
                number[pivot] = number[j];
                number[j] = temp;
                quicksort(number, first, j-1);
                quicksort(number, j+1, last);
        }
}

int main()
{
        int i, count, number[100];
        clrscr();
        printf("Enter the number of elements: ");
        scanf("%d",&count);

        printf("Enter %d elements:\n ", count);
        for(i=0;i<count;i++)
                scanf("%d",&number[i]);

        quicksort(number, 0, count - 1);

        printf("Order of Sorted elements after QuickSortS: \n");
        for(i=0; i<count; i++)
                printf(" %d",number[i]);

        getch();
        return 0;
}
```

QUICK SORT

```
#include<stdio.h>
#include<conio.h>
void quicksort(int number[25],int first,int last){
   int i, j, pivot, temp;

   if(first<last){
      pivot=first;
      i=first;
      j=last;

      while(i<j){
         while(number[i]<=number[pivot]&&i<last)
            i++;
         while(number[j]>number[pivot])
            j--;
         if(i<j){
            temp=number[i];
            number[i]=number[j];
            number[j]=temp;
         }
      }

      temp=number[pivot];
      number[pivot]=number[j];
      number[j]=temp;
      quicksort(number,first,j-1);
      quicksort(number,j+1,last);

   }
}

int main(){
   int i, count, number[25];

   printf("How many elements are u going to enter?: ");
   scanf("%d",&count);
```

```
   printf("Enter %d elements: ", count);
   for(i=0;i<count;i++)
      scanf("%d",&number[i]);

   quicksort(number,0,count-1);

   printf("Order of Sorted elements: ");
   for(i=0;i<count;i++)
      printf(" %d",number[i]);

   return 0;
}
```

✓ **Program 12 - Program to Traverse and insert an element in an array**

```
#include<stdio.h>
#include<conio.h>
int main()
{
        int i, n, num, pos, arr[10];
        clrscr();
```

```
	printf("\n Enter the number of elements in array:");
	scanf("%d",&n);
	printf("\n Enter the elements in array:\n ");
	for(i=0;i<n;i++)
	{
		printf("arr[%d]= ",i);
		scanf("%d",&arr[i]);
	}
	printf("\n Traversing the array:",i);
	for(i=0;i<n;i++)
	{
		printf("\n arr[%d] = %d", i, arr[i]);
	}
	getch();
	printf("\n Enter the number to be inserted:");
	scanf("%d", &num);
	printf("\n Enter the position at which the number has to be inserted:");
	scanf("%d",&pos);
	for(i=n-1;i>=pos;i--)
	{
		arr[i+1]=arr[i];
	}
	arr[pos] = num;
	n = n+1;
	printf("\n The array after insertion of %d is: ",num);
	for(i=0;i<n;i++)
	{
		printf("\n arr[%d] = %d", i, arr[i]);
	}
	getch();
	return 0;
}
```

✓ **Program 13 - Program to traverse and delete an element in an array**

```
#include<stdio.h>
#include<conio.h>
int main()
{
	int i, n, pos, arr[100];
	clrscr();
	printf("\n Enter the number of elements in array: ");
	scanf("%d",&n);
	printf("\n Enter the elements in array: ");
	for(i=0;i<n;i++)
	{
		printf("\n arr[%d] = ",i);
		scanf("%d",&arr[i]);
```

```
	}
	printf("Traversing the array",i);
	for(i=0;i<n;i++)
	{
		printf("\n arr[%d] = %d", i, arr[i]);
	}
	printf("\n Enter the position at which the element will be deleted:");
	scanf("%d",&pos);
	for(i=pos;i<n-1;i++)
	{
		arr[i] = arr[i+1];
		n--;
	}
	printf("\n The array after deletion is: ");
	for(i=0;i<n;i++)
	{
		printf("\n arr[%d] = %d", i, arr[i]);
	}
	getch();
	return 0;
}
```

- ✓ **Program 14: Program to traverse and transpose the 2D Array**

```
#include<stdio.h>
#include<conio.h>
int main()
{
	int a[5][5],trans[5][5],r,c,i,j,matrix[10][10];
	clrscr();
	printf("Enter rows and columns: ");
	scanf("%d %d",&r,&c);
	printf("\n Enter matrix element: \n");
	for(i=0;i<r;i++)
	{
		for(j=0;j<c;j++)
		{
			printf("Enter element a%d%d :",i+1,j+1);
			scanf("\n %d",&a[i][j]);
		}
	}
	printf("\n Traverse Matrix: \n");
	for(i=0;i<r;i++)
	{
		for(j=0;j<c;j++)
		{
			printf("%d ",a[i][j]);
			if(j==c-1)
```

```
                                printf("\n");
                }
        }
        for(i=0;i<r;i++)
        {
                for(j=0;j<c;j++)
                {
                        trans[j][i]=a[i][j];
                }
        }
        printf("\n Transpose of matrix: \n");
        for(i=0;i<c;i++)
        {
                for(j=0;j<r;j++)
                {
                        printf("%d ",trans[i][j]);
                        if(j==r-1)
                                printf("\n");
                }
        }
        getch();
        return 0;
}
```

✓ **Program 15: To create and diaplay Singly Linked List**

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

//Structure for a node in a linked list
struct node
{
        int num;            //Data of the node
        struct node *nextptr;   //Address of the next node
} *stnode;                  //Pointer to the starting node

//Function prototypes
void createNodeList(int n);        //Function to create the linked list
void displayList();                //Function to display the linked list

//Function to create a linked list with  a node
void createNodeList(int n)
{
        struct node *fnNode, *tmp;
        int num, i;

        //Allocating memory for the starting node
```

```
        stnode = (struct node *)malloc(sizeof(struct node));

        //Checking if memory allocation is successful
        if(stnode == NULL)
        {
                printf("Memory can not be allocated.");
        }
        else
        {
                //Reading data for the starting node from user input
                printf("Input data for node 1:");
                scanf("%d", &num);
                stnode->num = num;
                stnode->nextptr = NULL; //Setting the next pointer to NULL
                tmp = stnode;

                //Creating n nodes and adding them to the linked list
                for(i=2; i<=n; i++)
                {
                        fnNode = (struct node *)malloc(sizeof(struct node));

                        //Checking if memory allocation is successful
                        if(fnNode == NULL)
                        {
                                printf("Memory can not be allocated.");
                                break;
                        }
                        else
                        {
                                //Reading data for each node from user input
                                printf("Input data for node %d:", i);
                                scanf("%d", &num);

                                fnNode->num = num;     //Setting the data for fnNode
                                fnNode->nextptr = NULL; //Setting the next pointer to NULL

                                tmp->nextptr = fnNode;  //Linking the current node to fnNode
                                tmp = tmp->nextptr;      //Moving tmp to the next node
                        }
                }
        }

}

//Function to display the linked list
void displayList()
{
        struct node *tmp;
```

```
	if(stnode == NULL)
	{
		printf("List is empty:");
	}
	else
	{
		tmp = stnode;

		//Traversing the linked list and printing each node's data
		while(tmp != NULL)
		{
			printf("Data = %d\n", tmp->num);	//Printing the data of the current
node
			tmp = tmp->nextptr;			//Moving to the next node in the list
		}
	}
}

//Main Function
int main()
{
	int n;
	clrscr();

	//Displaying the purpose of the program
	printf("\n\n Linked List: To create and display Singly Linked List: \n");
	printf("------------------------------------------------------------\n");

	//Inputting the number of nodes for the linked list
	printf("Input the number of nodes:");
	scanf("%d", &n);

	//Creating the linked list with n nodes
	createNodeList(n);

	//Displaying the data entered in the Linked List
	printf("Daata entered in the list: \n");

	displayList();
	getch();
```

✓ **Program 16: Insert a node at the beginning of the Singly Linked List**

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

//Structure for a node in a linked list
```

```
struct node
{
	int num;		//Data of the node
	struct node *nextptr;	//Address of the next node
} *stnode;			//Pointer to the starting node

//Function prototypes
void createNodeList(int n);	//Function to create the list
void NodeInsertatBegin(int num);	//Function to cinsert node at the begining of the linked list
void displayList();			//Function to display the linked list

//Function to create a linked list with  a node
void createNodeList(int n)
{
	struct node *fnNode, *tmp;
	int num, i;

	//Allocating memory for the starting node
	stnode = (struct node *)malloc(sizeof(struct node));

	//Checking if memory allocation is successful
	if(stnode == NULL)
	{
		printf("Memory can not be allocated.");
	}
	else
	{
		//Reading data for the starting node from user input
		printf("Input data for node 1:");
		scanf("%d", &num);
		stnode->num = num;
		stnode->nextptr = NULL; //Setting the next pointer to NULL
		tmp = stnode;

		//Creating n nodes and adding them to the linked list
		for(i=2; i<=n; i++)
		{
			fnNode = (struct node *)malloc(sizeof(struct node));

			//Checking if memory allocation is successful
			if(fnNode == NULL)
			{
				printf("Memory can not be allocated.");
				break;
			}
			else
			{
				//Reading data for each node from user input
```

```
                                printf("Input data for node %d:", i);
                                scanf("%d", &num);

                                fnNode->num = num;    //Setting the data for fnNode
                                fnNode->nextptr = NULL; //Setting the next pointer to NULL

                                tmp->nextptr = fnNode;  //Linking the current node to fnNode
                                tmp = tmp->nextptr;     //Moving tmp to the next node
                        }
                }
        }

}

void NodeInsertatBegin(int num)
{
        struct node *fnNode;
        fnNode = (struct node*)malloc(sizeof(struct node));
        if(fnNode == NULL)
        {
                printf("Memory can not be allocated.");
        }
        else
        {
                fnNode->num = num;    //links the data part
                fnNode->nextptr = stnode;       //links the address part
                stnode = fnNode;          //Makes stnode as first node
        }
}
//Function to display the linked list
void displayList()
{
        struct node *tmp;
        if(stnode == NULL)
        {
                printf("List is empty:");
        }
        else
        {
                tmp = stnode;

                //Traversing the linked list and printing each node's data
                while(tmp != NULL)
                {
                        printf("Data = %d\n", tmp->num);        //Printing the data of the current
node
                        tmp = tmp->nextptr;                     //Moving to the next node in the list
                }
```

```
        }
}

//Main Function
int main()
{
        int n, num;
        clrscr();

        //Displaying the purpose of the program
        printf("\n\n Linked List: To create and display Singly Linked List: \n");
        printf("-------------------------------------------------------------\n");

        //Inputting the number of nodes for the linked list
        printf("Input the number of nodes:");
        scanf("%d", &n);

        //Creating the linked list with n nodes
        createNodeList(n);

        //Displaying the data entered in the Linked List
        printf("Daata entered in the list: \n");
        displayList();

        printf("\n Input data to insert at the beginning of the list:");
        scanf("%d", &num);

        NodeInsertatBegin(num);
        printf("\n Data after inserted in the list are: \n");
        displayList();
        getch();
        return 0;
}
```

// Inserting a node in single linked list at given position.

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

// Define the structure for a node
struct Node {
   int data;
   struct Node* next;
};

// Function to create a new node
struct Node* createNode(int data) {
   struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
   newNode->data = data;
   newNode->next = NULL;
   return newNode;
}

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

// Function to add a node at a specific position
```

```
void addNodeAtPosition(struct Node** head, int data, int position) {
    // Create a new node
    struct Node* newNode = createNode(data);

    // If position is 0, insert at the beginning
    if (position == 0) {
        newNode->next = *head;
        *head = newNode;
        return;
    }

    // Traverse the list to find the (position-1)th node
    struct Node* temp = *head;
    for (int i = 0; temp != NULL && i < position - 1; i++) {
        temp = temp->next;
    }

    // If the position is greater than the number of nodes, print an error
    if (temp == NULL) {
        printf("Position out of range!\n");
        free(newNode);
        return;
    }

    // Insert the new node at the desired position
    newNode->next = temp->next;
    temp->next = newNode;
}
```

```
int main() {
    struct Node* head = NULL;

    // Adding nodes to the list
    addNodeAtPosition(&head, 10, 0); // Add 10 at position 0
    addNodeAtPosition(&head, 20, 1); // Add 20 at position 1
    addNodeAtPosition(&head, 30, 1); // Add 30 at position 1
    addNodeAtPosition(&head, 40, 5); // Position out of range (invalid)

    // Print the list
    printList(head);

    return 0;
}
```

OUTPUT:

10 -> 30 -> 20 -> NULL

// Delete a front node from Single Linked List

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

// Define the structure for a linked list node
struct Node {
    int data;
    struct Node *next;
};

// Function to delete the front node (head) from the linked list
void deleteFrontNode(struct Node **head) {
```

```
    // Check if the list is empty
    if (*head == NULL) {
        printf("List is empty, nothing to delete.\n");
        return;
    }

    // Store the current head node
    struct Node *temp = *head;

    // Move the head pointer to the next node
    *head = (*head)->next;

    // Free the memory of the old head node
    free(temp);

    printf("Front node deleted successfully.\n");
}

// Function to insert a new node at the beginning of the list
void insertFront(struct Node **head, int data) {
    struct Node *newNode = (struct Node *)malloc(sizeof(struct Node));
    newNode->data = data;
    newNode->next = *head;
    *head = newNode;
}

// Function to print the linked list
void printList(struct Node *head) {
    if (head == NULL) {
```

```
        printf("List is empty.\n");
        return;
    }

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

int main() {
    struct Node *head = NULL;

    // Inserting some nodes at the front of the list
    insertFront(&head, 10);
    insertFront(&head, 20);
    insertFront(&head, 30);
    insertFront(&head, 40);

    printf("Original Linked List: ");
    printList(head);

    // Deleting the front node
    deleteFrontNode(&head);

    printf("Linked List after deleting the front node: ");
    printList(head);
```

```
    return 0;
}
```

OUTPUT:

Original Linked List: 40 -> 30 -> 20 -> 10 -> NULL

Front node deleted successfully.

Linked List after deleting the front node: 30 -> 20 -> 10 -> NULL

//Delete a node from given position from Single linked List

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

// Definition of a single linked list node
struct Node {
    int data;
    struct Node* next;
};

// Function to create a new node
struct Node* createNode(int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = data;
    newNode->next = NULL;
    return newNode;
}

// Function to insert a node at the end of the linked list
void append(struct Node** head, int data) {
    struct Node* newNode = createNode(data);
```

```
    if (*head == NULL) {
        *head = newNode;
        return;
    }
    struct Node* temp = *head;
    while (temp->next != NULL) {
        temp = temp->next;
    }
    temp->next = newNode;
}

// Function to delete a node at a given position (1-based index)
void deleteNode(struct Node** head, int position) {
    if (*head == NULL) {
        printf("List is empty.\n");
        return;
    }

    struct Node* temp = *head;

    // If the position to be deleted is the head node
    if (position == 1) {
        *head = temp->next;  // Change the head
        free(temp);  // Free the old head node
        return;
    }

    // Find the previous node of the node to be deleted
    for (int i = 1; temp != NULL && i < position - 1; i++) {
```

```
        temp = temp->next;
    }

    // If the position is more than the number of nodes
    if (temp == NULL || temp->next == NULL) {
        printf("Position is out of range.\n");
        return;
    }

    // Node temp->next is the node to be deleted
    struct Node* nodeToDelete = temp->next;
    temp->next = temp->next->next;  // Unlink the node from the list
    free(nodeToDelete);  // Free the memory of the node
}

// Function to print the linked list
void printList(struct Node* head) {
    if (head == NULL) {
        printf("List is empty.\n");
        return;
    }
    struct Node* temp = head;
    while (temp != NULL) {
        printf("%d -> ", temp->data);
        temp = temp->next;
    }
    printf("NULL\n");
}
```

```
// Main function
int main() {
    struct Node* head = NULL;

    // Append some nodes to the list
    append(&head, 10);
    append(&head, 20);
    append(&head, 30);
    append(&head, 40);
    append(&head, 50);

    printf("Original List: ");
    printList(head);

    int position;
    printf("Enter position to delete node: ");
    scanf("%d", &position);

    // Delete node at the given position
    deleteNode(&head, position);

    printf("List after deletion: ");
    printList(head);

    return 0;
}
```

OUTPUT:

Original List: 10 -> 20 -> 30 -> 40 -> 50 -> NULL

Enter position to delete node: 3

List after deletion: 10 -> 20 -> 40 -> 50 -> NULL

//Reversing a single linked List

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

// Define the node structure
struct Node {
    int data;
    struct Node* next;
};

// Function to reverse the linked list
void reverse(struct Node** head) {
    struct Node* prev = NULL;
    struct Node* current = *head;
    struct Node* next = NULL;

    while (current != NULL) {
        next = current->next;  // Save next node
        current->next = prev;  // Reverse the current node's pointer
        prev = current;        // Move prev and current one step forward
        current = next;
    }
    *head = prev;  // Update head to the new first node
}

// Function to print the linked list
void printList(struct Node* head) {
```

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

// Function to add a node at the beginning
void push(struct Node** head, int new_data) {
    struct Node* new_node = (struct Node*)malloc(sizeof(struct Node));
    new_node->data = new_data;
    new_node->next = *head;
    *head = new_node;
}

int main() {
    struct Node* head = NULL;

    push(&head, 10);
    push(&head, 20);
    push(&head, 30);
    push(&head, 40);

    printf("Original List: ");
    printList(head);

    reverse(&head);
```

```
    printf("Reversed List: ");
    printList(head);

    return 0;
}
```

OUTPUT:

Original Linked List:

1 -> 2 -> 3 -> 4 -> 5 -> NULL

Reversed Linked List:

5 -> 4 -> 3 -> 2 -> 1 -> NULL

//Concatenation of two single linked list

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

// Define a node structure for singly linked list
struct Node {
    int data;
    struct Node* next;
};

// Function to create a new node
struct Node* createNode(int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = data;
    newNode->next = NULL;
    return newNode;
}
```

```
// Function to append a node at the end of the list
void append(struct Node** head, int data) {
    struct Node* newNode = createNode(data);
    if (*head == NULL) {
        *head = newNode;
    } else {
        struct Node* temp = *head;
        while (temp->next != NULL) {
            temp = temp->next;
        }
        temp->next = newNode;
    }
}

// Function to concatenate two linked lists
void concatenate(struct Node** head1, struct Node** head2) {
    if (*head1 == NULL) {
        *head1 = *head2;
        return;
    }

    struct Node* temp = *head1;
    while (temp->next != NULL) {
        temp = temp->next;
    }

    temp->next = *head2;
}
```

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

int main() {
    struct Node* list1 = NULL;
    struct Node* list2 = NULL;

    // Adding elements to list1
    append(&list1, 1);
    append(&list1, 2);
    append(&list1, 3);

    // Adding elements to list2
    append(&list2, 4);
    append(&list2, 5);

    printf("List 1 before concatenation: ");
    printList(list1);

    printf("List 2 before concatenation: ");
    printList(list2);
```

```
    // Concatenate list2 to list1
    concatenate(&list1, &list2);

    printf("List 1 after concatenation: ");
    printList(list1);

    return 0;
}
```

OUTPUT:

List 1 before concatenation: 1 -> 2 -> 3 -> NULL

List 2 before concatenation: 4 -> 5 -> NULL

List 1 after concatenation: 1 -> 2 -> 3 -> 4 -> 5 -> NULL

//Insert node at a given position in Double linked List

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

// Define the structure for a node in the doubly linked list
struct Node {
    int data;
    struct Node* next;
    struct Node* prev;
};

// Function to create a new node
struct Node* createNode(int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = data;
```

```
    newNode->next = NULL;
    newNode->prev = NULL;
    return newNode;
}

// Function to insert a node at a given position in the doubly linked list
void insertAtPosition(struct Node** head, int position, int data) {
    struct Node* newNode = createNode(data);

    // If the list is empty or inserting at the beginning (position 0)
    if (*head == NULL || position == 0) {
        newNode->next = *head;
        if (*head != NULL) {
            (*head)->prev = newNode;
        }
        *head = newNode;
        return;
    }

    struct Node* temp = *head;
    int currentPos = 0;

    // Traverse the list to the specified position
    while (temp != NULL && currentPos < position - 1) {
        temp = temp->next;
        currentPos++;
    }

    // If the position is beyond the end of the list, we can insert at the end
```

```
    if (temp == NULL) {
        printf("The position is beyond the end of the list. Inserting at the end.\n");
        temp = *head;
        while (temp->next != NULL) {
            temp = temp->next;
        }
    }

    // Insert the new node at the desired position
    newNode->next = temp->next;
    newNode->prev = temp;

    // Update the next node if it's not NULL
    if (temp->next != NULL) {
        temp->next->prev = newNode;
    }

    // Link the previous node to the new node
    temp->next = newNode;
}

// Function to print the list (forward traversal)
void printList(struct Node* head) {
    struct Node* temp = head;
    while (temp != NULL) {
        printf("%d <-> ", temp->data);
        temp = temp->next;
    }
    printf("NULL\n");
```

```
}

// Main function to test the insert function
int main() {
    struct Node* head = NULL;

    // Insert nodes at various positions
    insertAtPosition(&head, 0, 10); // Insert 10 at position 0
    insertAtPosition(&head, 1, 20); // Insert 20 at position 1
    insertAtPosition(&head, 1, 15); // Insert 15 at position 1
    insertAtPosition(&head, 3, 25); // Insert 25 at position 3

    // Print the list after insertions
    printList(head);

    return 0;
}
```

OUTPUT:

10 <-> 15 <-> 20 <-> 25 <-> NULL

//delete a node from given position in Double linked list

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

// Structure for a doubly linked list node
struct Node {
    int data;
    struct Node* prev;
    struct Node* next;
```

```
};

// Function to create a new node
struct Node* createNode(int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = data;
    newNode->prev = NULL;
    newNode->next = NULL;
    return newNode;
}

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

// Function to delete a node at a given position (1-based index)
void deleteNode(struct Node** head, int position) {
    if (*head == NULL || position <= 0) {
        printf("Invalid position or empty list.\n");
        return;
    }

    struct Node* temp = *head;
```

```
// If the node to be deleted is the head
if (position == 1) {
    *head = temp->next;  // Change head to next node
    if (*head != NULL) {
        (*head)->prev = NULL;
    }
    free(temp);
    return;
}

// Traverse the list to find the node at the given position
for (int i = 1; temp != NULL && i < position; i++) {
    temp = temp->next;
}

// If position is greater than the number of nodes
if (temp == NULL) {
    printf("Position out of range.\n");
    return;
}

// Remove the node from the list
if (temp->next != NULL) {
    temp->next->prev = temp->prev;
}
if (temp->prev != NULL) {
    temp->prev->next = temp->next;
}
```

```
    free(temp);
}

// Main function to test the deletion
int main() {
    struct Node* head = createNode(1);
    struct Node* second = createNode(2);
    struct Node* third = createNode(3);
    struct Node* fourth = createNode(4);

    // Linking nodes
    head->next = second;
    second->prev = head;
    second->next = third;
    third->prev = second;
    third->next = fourth;
    fourth->prev = third;

    printf("Original List: ");
    printList(head);

    // Delete node at position 3 (1-based index)
    deleteNode(&head, 3);

    printf("List after deletion at position 3: ");
    printList(head);

    return 0;
```

```
}
```

OUTPUT:

Original List: 1 2 3 4

List after deletion at position 3: 1 2 4

//Implementation of Push and Pop Operation on Stack using Array

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

#define MAX 5  // Maximum size of the stack

// Stack structure
struct Stack {
   int arr[MAX];  // Array to hold stack elements
   int top;       // Index of the top element
};

// Function to initialize the stack
void initialize(struct Stack* stack) {
   stack->top = -1;  // Stack is initially empty
}

// Function to check if the stack is full
int isFull(struct Stack* stack) {
   return stack->top == MAX - 1;
}

// Function to check if the stack is empty
int isEmpty(struct Stack* stack) {
```

```
    return stack->top == -1;
}

// Function to push an element onto the stack
void push(struct Stack* stack, int value) {
    if (isFull(stack)) {
        printf("Stack Overflow! Cannot push %d\n", value);
    } else {
        stack->arr[++(stack->top)] = value;  // Increment top and add the element
        printf("%d pushed onto stack\n", value);
    }
}

// Function to pop an element from the stack
int pop(struct Stack* stack) {
    if (isEmpty(stack)) {
        printf("Stack Underflow! Cannot pop\n");
        return -1;  // Return -1 if the stack is empty
    } else {
        int poppedValue = stack->arr[stack->top--];  // Return the top element and decrement top
        return poppedValue;
    }
}

// Function to display the stack elements
void display(struct Stack* stack) {
    if (isEmpty(stack)) {
        printf("Stack is empty\n");
    } else {
```

```
        printf("Stack elements: ");
        for (int i = stack->top; i >= 0; i--) {
            printf("%d ", stack->arr[i]);
        }
        printf("\n");
    }
}

int main() {
    struct Stack stack;  // Declare a stack
    initialize(&stack);  // Initialize the stack

    // Perform some push operations
    push(&stack, 10);
    push(&stack, 20);
    push(&stack, 30);

    // Display the stack
    display(&stack);

    // Perform some pop operations
    printf("Popped value: %d\n", pop(&stack));
    printf("Popped value: %d\n", pop(&stack));

    // Display the stack again
    display(&stack);

    // Attempting to pop from an empty stack
    pop(&stack);
```

```
    pop(&stack);

    return 0;
}
```

OUTPUT:

```
10 pushed onto stack
20 pushed onto stack
30 pushed onto stack
Stack elements: 30 20 10
Popped value: 30
Popped value: 20
Stack elements: 10
Stack Underflow! Cannot pop
Stack Underflow! Cannot pop
```

//Evaluate postfix Expression using Stack

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

#define MAX 100

// Stack structure
struct Stack {
    int arr[MAX];
    int top;
};

// Function to initialize the stack
```

```
void initStack(struct Stack *s) {
    s->top = -1;
}

// Function to push an element to the stack
void push(struct Stack *s, int value) {
    if (s->top == MAX - 1) {
        printf("Stack Overflow\n");
        return;
    }
    s->arr[++(s->top)] = value;
}

// Function to pop an element from the stack
int pop(struct Stack *s) {
    if (s->top == -1) {
        printf("Stack Underflow\n");
        return -1;
    }
    return s->arr[(s->top)--];
}

// Function to evaluate a postfix expression
int evaluatePostfix(char *exp) {
    struct Stack s;
    initStack(&s);
    int i = 0;

    while (exp[i] != '\0') {
```

```
        // If the character is a digit, push it to the stack
        if (isdigit(exp[i])) {
            push(&s, exp[i] - '0');
        }
        // If the character is an operator, pop two elements, perform the operation and push the result
        else if (exp[i] == '+' || exp[i] == '-' || exp[i] == '*' || exp[i] == '/') {
            int operand2 = pop(&s);
            int operand1 = pop(&s);
            int result;

            switch (exp[i]) {
                case '+': result = operand1 + operand2; break;
                case '-': result = operand1 - operand2; break;
                case '*': result = operand1 * operand2; break;
                case '/': result = operand1 / operand2; break;
                default:
                    printf("Invalid operator\n");
                    return -1;
            }
            push(&s, result);
        }
        i++;
    }

    // The result will be the last element in the stack
    return pop(&s);
}

int main() {
```

```
    char exp[MAX];
    printf("Enter a postfix expression: ");
    scanf("%s", exp);

    int result = evaluatePostfix(exp);
    printf("Result: %d\n", result);

    return 0;
}
```

OUTPUT:

Enter a postfix expression: 23*54*+9-

Result: 17

//Insert elements in a Circular Queue using array

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

#define MAX 5  // Maximum size of the Circular Queue

// Circular Queue structure
struct CircularQueue {
    int arr[MAX];
    int front, rear;
};

// Function to initialize the queue
void initQueue(struct CircularQueue* queue) {
    queue->front = -1;
    queue->rear = -1;
```

```
}

// Function to check if the queue is full
int isFull(struct CircularQueue* queue) {
    return ((queue->rear + 1) % MAX == queue->front);
}

// Function to check if the queue is empty
int isEmpty(struct CircularQueue* queue) {
    return (queue->front == -1);
}

// Function to insert an element into the circular queue
void enqueue(struct CircularQueue* queue, int value) {
    if (isFull(queue)) {
        printf("Queue is full! Cannot insert %d\n", value);
    } else {
        if (queue->front == -1) {  // Queue is empty
            queue->front = 0;
        }
        queue->rear = (queue->rear + 1) % MAX;  // Circular increment
        queue->arr[queue->rear] = value;
        printf("Inserted %d into the queue\n", value);
    }
}

// Function to display the queue
void displayQueue(struct CircularQueue* queue) {
    if (isEmpty(queue)) {
```

```
        printf("Queue is empty!\n");
        return;
    }
    printf("Queue contents: ");
    int i = queue->front;
    while (i != queue->rear) {
        printf("%d ", queue->arr[i]);
        i = (i + 1) % MAX;  // Circular increment
    }
    printf("%d\n", queue->arr[queue->rear]);  // Print the last element
}

int main() {
    struct CircularQueue queue;
    initQueue(&queue);

    // Insert elements into the queue
    enqueue(&queue, 10);
    enqueue(&queue, 20);
    enqueue(&queue, 30);
    enqueue(&queue, 40);
    enqueue(&queue, 50);

    // Display the queue
    displayQueue(&queue);

    // Try to insert when the queue is full
    enqueue(&queue, 60);
```

```
    // Display the queue after insertion
    displayQueue(&queue);

    return 0;
}
```

OUTPUT:

```
Inserted 10 into the queue
Inserted 20 into the queue
Inserted 30 into the queue
Inserted 40 into the queue
Inserted 50 into the queue
Queue contents: 10 20 30 40 50
Queue is full! Cannot insert 60
Queue contents: 10 20 30 40 50
```

// delete element from Circular Queue using array

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

#define MAX 5  // Maximum size of the queue

// Circular Queue structure
typedef struct {
    int arr[MAX];
    int front;
    int rear;
} CircularQueue;

// Function to initialize the queue
```

```
void initializeQueue(CircularQueue* q) {
    q->front = -1;
    q->rear = -1;
}

// Function to check if the queue is empty
int isEmpty(CircularQueue* q) {
    return (q->front == -1);
}

// Function to check if the queue is full
int isFull(CircularQueue* q) {
    return ((q->rear + 1) % MAX == q->front);
}

// Function to enqueue an element to the queue
void enqueue(CircularQueue* q, int value) {
    if (isFull(q)) {
        printf("Queue is full! Cannot enqueue %d\n", value);
    } else {
        if (q->front == -1) {  // If the queue is empty
            q->front = 0;
        }
        q->rear = (q->rear + 1) % MAX;  // Circular increment
        q->arr[q->rear] = value;
        printf("%d enqueued to queue\n", value);
    }
}
```

```
// Function to dequeue an element from the queue
void dequeue(CircularQueue* q) {
    if (isEmpty(q)) {
        printf("Queue is empty! Cannot dequeue.\n");
    } else {
        int dequeuedValue = q->arr[q->front];
        if (q->front == q->rear) {  // Only one element in the queue
            q->front = -1;
            q->rear = -1;
        } else {
            q->front = (q->front + 1) % MAX;  // Circular increment
        }
        printf("%d dequeued from queue\n", dequeuedValue);
    }
}

// Function to display the elements of the queue
void displayQueue(CircularQueue* q) {
    if (isEmpty(q)) {
        printf("Queue is empty.\n");
    } else {
        int i = q->front;
        printf("Queue elements: ");
        while (i != q->rear) {
            printf("%d ", q->arr[i]);
            i = (i + 1) % MAX;
        }
        printf("%d\n", q->arr[q->rear]);
    }
```

```
}

int main() {
    CircularQueue q;
    initializeQueue(&q);

    enqueue(&q, 10);
    enqueue(&q, 20);
    enqueue(&q, 30);
    enqueue(&q, 40);
    enqueue(&q, 50);

    displayQueue(&q);

    dequeue(&q);  // Remove the front element
    displayQueue(&q);

    enqueue(&q, 60);  // Add a new element after deletion
    displayQueue(&q);

    dequeue(&q);  // Remove another front element
    displayQueue(&q);

    return 0;
}
```

OUTPUT:

```
10 enqueued to queue
20 enqueued to queue
30 enqueued to queue
```

40 enqueued to queue

50 enqueued to queue

Queue elements: 10 20 30 40 50

10 dequeued from queue

Queue elements: 20 30 40 50

60 enqueued to queue

Queue elements: 20 30 40 50 60

20 dequeued from queue

Queue elements: 30 40 50 60