1)**Write a C Program that computes the real roots of a quadratic function. Your program should begin by prompting the user for the values of a, b and c. Then it should display a message indicating the nature of real roots, along with the values of the real roots (if any).**

#include<stdio.h>

#include<math.h>

int main()

{

float a,b,c,dis,root1,root2;

printf("Enter coefficient of x2 :");

scanf("%f",&a);

printf("Enter the coefficient of x : ");

scanf("%f",&b);

printf("Enter the value of constant :");

scanf("%f",&c);

root1=((-b)+pow((b\*b)-(4\*a\*c),0.5))/(2\*a);

root2=((-b)-pow((b\*b)-(4\*a\*c),0.5))/(2\*a);

printf("The roots of the quadratic equation is : %f and %f ",root1,root2);

dis=(b\*b)-(4\*a\*c);

if(dis<0){

printf("\nThe roots are imaginary");}

else if(dis>0){

printf("\nThe roots are real");}

else if(root1==root2){

printf("\nThe roots are equal");}

return 0;

}

2) **Write a C Program to input percentage of marks and display grade according to**

**following:**

**Percentage >= 90% : Grade A**

**Percentage>= 80% : Grade B**

**Percentage >= 70% : Grade C**

**Percentage >= 60% : Grade D**

**Percentage >= 40% : Grade E**

**Percentage >= 40% : Grade F**

#include<stdio.h>

void main()

{

int marks;

printf("Enter your marks ");

scanf("%d", &marks);

if(marks<40 && marks>=0)

{

printf("Grade F");

}

else if(marks>=40 && marks<60)

{

printf("Grade E");

}

else if(marks>=60 && marks<70)

{

printf("Grade D");

}

else if(marks>=70 && marks<80)

{

printf("Grade C");

}

else if(marks>=80 && marks<90)

{

printf("Grade B");

}

else if(marks>=90 && marks<=100)

{

printf("Grade A");

}

else

{

printf("enter correct marks");

}

}

3) **Write a C Program to Add, Subtract, Multiply or Divide Using switch...case**

**(menu driven).**

#include<stdio.h>

int main()

{

int c;

int num1,num2;

printf("Enter value of first number:");

scanf("%d",&num1);

printf("Enter the value of second number :");

scanf("%d",&num2);

printf("\nEnter\*1\* for addition,\*2\* for subtraction,\*3\* for multiplication,\*4\* for division");

printf("\nEnter the operation to be done :");

scanf("%d",&c);

switch(c)

{

case 1:

printf("You have chosen to add two numbers ");

printf("\nThe addition of two numbers is %d",num1+num2);

break;

case 2:

printf(" You ahve chosen to subtract two numbers ");

printf("\nThe difference of two numbers is %d",num1-num2);

break;

case 3:

printf(" You have chosen to multiply two numbers");

printf("\n The product of two numbers is %d",num1\*num2);

break;

case 4:

printf(" You have chosen to divide two numbers");

printf("\nThe division of two numbers is %d",num1/num2);

break;

default:

printf("\nEnter appropriate number for operation");

}

return 0;

}

4) **Write a C Program to find the largest of three numbers using a conditional**

**operator.**

# include <stdio.h>

void main()

{

int first, second, third, largest ;

printf("Enter the three numbers : ") ;

scanf("%d %d %d", &first, &second, &third) ;

largest = first > second ? (first > third ? first : third) : (second > third ? second : third) ;

printf("\n The largest number is : %d", largest) ;

}

5) **Write a C Program to Check Whether a Character is Vowel or not using switch**

**case.**

#include<stdio.h>

int main()

{

char ch;

printf("\n Enter a character to check whether it is vowel or consonant :");

scanf("%c",&ch);

switch(ch)

{

case 'A':

case 'a':

printf("\n %c is a vowel",ch);

break;

case 'E':

case 'e':

printf("\n %c is a vowel",ch);

break;

case 'I':

case 'i':

printf("\n %c is a vowel",ch);

break;

case 'O':

case 'o':

printf("\n %c is a vowel",ch);

break;

case 'U':

case 'u':

printf("\n %c is a vowel",ch);

break;

default:

printf("\n %c is not a vowel",ch);

}

return 0;

}

6)**Write a C Program to calculate factorial of a number.**

#include<stdio.h>

void main()

{

int num ,j=1,fact=1;

printf("\n Enter The value of num:");

scanf("%d", &num);

do

{

fact=fact\*j;

j++;

}while(j<=num);

printf("\n The Factorial of %d is %d", num, fact);

}

7) **Write a C Program to check if the number given by the user is prime or not.**

#include<stdio.h>

int main()

{

int i,j;

for(i=1;i<5;i++)

{

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

printf("\*");

}

printf("\n");

}

return 0;

}

8) **WAP to print the following structure**

**\***

**\*\***

**\*\*\***

**\* \* \* \***

#include <stdio.h>

int main()

{

int i, j;

for(i=1;i<=4;i++)

{

printf("\n");

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

printf("\*");

}

return 0;

}

9) **Write a C Program to print the following structure**

**1**

**1 2**

**1 2 3**

**1 2 3 4**

#include<stdio.h>

int fibo(int);

int main()

{

int num,i=0,result;

printf("Enter the number for which fibonacci's should be given :");

scanf("%d", &num);

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

{

result= fibo(i);

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

}

return 0;

}

int fibo(int a)

{

if(a==0)

{

return 0;

}

else if(a==1)

{

return 1;

}

else

{

return ( fibo(a-1) + fibo(a-2));

}

}

10) **Write a C Program to Display Fibonacci Series**

#include<stdio.h>

int main ()

{

int a = 0, b = 1, c, i, n;

printf ("Enter the length of the Fibonacci series \n");

scanf ("%d", &n);

printf ("The Fibonacci series is :\n");

printf ("%d%d", a, b);

for (i = 2; i <= n; i++)

{

c = a + b;

printf ("%d ", third);

first = second;

second = third;

}

return 0;

}

11) **Write a C Program to calculate Sum &amp; Average of an Array.**

#include<stdio.h>

int main()

{

int arr[1000],num,sum=0,i,j;

float avg;

printf("Enter the number of elements to be in the array :");

scanf("%d",&num);

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

{

printf("Enter the number :");

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

}

printf("The array is...");

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

{

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

}

for(j=0;j<num;j++)

{

sum+=arr[j];

}

avg=sum/num;

printf("\nThe sum of the numbers is %d",sum);

printf("\nThe average of the numbers is %.2lf",avg);

return 0;

}

12) **Write a C Program to Find the Largest number in a given Array and its index.**

#include <stdio.h>

int main()

{

int array[100], maximum, size, c, location = 1;

printf("Enter the number of elements in array\n");

scanf("%d", &size);

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

for (c = 0; c < size; c++)

scanf("%d", &array[c]);

maximum = array[0];

for (c = 1; c < size; c++)

{

if (array[c] > maximum)

{

maximum = array[c];

location = c+1;

}

}

printf("Maximum element is present at location %d and its value is %d.\n", location, maximum);

return 0;

}

13) **Write a C Program to search for a number in the one dimensional array using a linear search algorithm**

#include<stdio.h>

int main()

{

int arr[1000],i,num,find,result=0;

printf("Enter the number of elements :");

scanf("%d",&num);

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

{

printf("Enter the number :");

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

}

printf("Enter the number to be found :");

scanf("%d",&find);

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

{

if(arr[i]==find)

{

printf("The number is found in the array");

}

}

return 0;

}

14) **Write a C Program for Binary search**

#include <stdio.h>

int main()

{

int c, first, last, middle, n, search, array[100];

printf("Enter number of elements\n");

scanf("%d", &n);

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

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

scanf("%d", &array[c]);

printf("Enter value to find\n");

scanf("%d", &search);

first = 0;

last = n - 1;

middle = (first+ last)/2;

while (first <= last) {

if (array[middle] < search)

first = middle + 1;

else if (array[middle] == search) {

printf("%d found at location %d.\n", search, middle+1);

break;

}

else

last = middle - 1;

middle = (first + last)/2;

}

if (first > last)

printf("Not found! %d isn't present in the list.\n", search);

return 0;

}

**15)** Write a C Program to Sort the Array in an Ascending Order using Bubble sort.

#include<stdio.h>

int main()

{

int arr[1000],i,j,k,temp,num;

printf("Enter the number of elements to be in the array :");

scanf("%d",&num);

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

{

printf("Enter the element :");

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

}

printf("The array is.....");

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

{

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

}

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

{

for(j=0;j<num-i-1;j++)

{

if(arr[j]>arr[j+1])

{

temp=arr[j];

arr[j]=arr[j+1];

arr[j+1]=temp;

}

}

}

printf("The sorted array is...");

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

{

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

}

return 0;

}

16) **Write a C Program to sort an array in descending order using Selection sort.**

#include <stdio.h>

int main()

{

int Array[50], i, j, temp, Size;

printf("\n Enter the Number of elements in an array : ");

scanf("%d", &Size);

printf("\n Enter %d elements of an Array \n", Size);

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

{

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

}

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

{

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

{

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

{

temp = Array[i];

Array[i] = Array[j];

Array[j] = temp;

}

}

}

printf("\n \*\*\*\* Array of Elements in Descending Order are : \*\*\*\*\n");

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

{

printf("%d\t", Array[i]);

}

return 0;

}

**17) Write a C Program to sort an array in ascending order using Insertion sort**.

#include <stdio.h>

int main()

{

int n, array[100], c, d, t, flag = 0;

printf("Enter number of elements needed: \n");

scanf("%d", &n);

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

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

scanf("%d", &array[c]);

for (c = 1 ; c <= n - 1; c++) {

t = array[c];

for (d = c - 1 ; d >= 0; d--) {

if (array[d] > t) {

array[d+1] = array[d];

flag = 1;

}

else

break;

}

if (flag)

array[d+1] = t;

}

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

for (c = 0; c <= n - 1; c++) {

printf("%d\n", array[c]);

}

return 0;

}

18) **Write a C Program to sort an array in descending order using Heap sort.**

#include <bits/stdc++.h>

using namespace std;

void heapify(int arr[], int n, int i)

{

int smallest = i;

int l = 2 \* i + 1;

int r = 2 \* i + 2;

if (l < n && arr[l] < arr[smallest])

smallest = l;

if (r < n && arr[r] < arr[smallest])

smallest = r;

if (smallest != i) {

swap(arr[i], arr[smallest]);

heapify(arr, n, smallest);

}

}

void heapSort(int arr[], int n)

{

for (int i = n / 2 - 1; i >= 0; i--)

heapify(arr, n, i);

for (int i = n - 1; i >= 0; i--) {

swap(arr[0], arr[i]);

heapify(arr, i, 0);

}

}

void printArray(int arr[], int n)

{

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

cout << arr[i] << " ";

cout << "\n";

}

int main()

{

int arr[] = { 4, 6, 3, 2, 9 };

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

heapSort(arr, n);

cout << "Sorted array is \n";

printArray(arr, n);

}

**19)Write a C program that takes an array and returns a new array with unique**

**elements of the first array.**

**#include <stdio.h>**

**#define MAX\_SIZE 100**

**int main()**

**{**

**int arr[MAX\_SIZE], freq[MAX\_SIZE];**

**int size, i, j, count;**

**printf("Enter size of array: ");**

**scanf("%d", &size);**

**printf("Enter elements in array: ");**

**for(i=0; i<size; i++)**

**{**

**scanf("%d", &arr[i]);**

**freq[i] = -1;**

**}**

**for(i=0; i<size; i++)**

**{**

**count = 1;**

**for(j=i+1; j<size; j++)**

**{**

**if(arr[i] == arr[j])**

**{**

**count++;**

**freq[j] = 0;**

**}**

**}**

**if(freq[i] != 0)**

**{**

**freq[i] = count;**

**}**

**}**

**printf("\n Unique elements in the array are: ");**

**for(i=0; i<size; i++)**

**{**

**if(freq[i] == 1)**

**{**

**printf("%d ", arr[i]);**

**}**

**}**

**return 0;**

**}**

**20) Write a C Program to input two matrices of 5×5 add them and output the**

**resultant matrix.**

#include <stdio.h>

int main() {

int r, c, a[10][10], b[10][10], sum[10][10], i, j;

printf("Enter the number of rows : ");

scanf("%d", &r);

printf("Enter the number of columns : ");

scanf("%d", &c);

printf("\n Enter elements of 1st matrix:\n");

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

for (j = 0; j < c; ++j) {

printf("Enter element a%d%d: ", i + 1, j + 1);

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

}

printf("Enter elements of 2nd matrix:\n");

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

for (j = 0; j < c; ++j) {

printf("Enter element a %d%d: ", i + 1, j + 1);

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

}

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

for (j = 0; j < c; ++j) {

sum[i][j] = a[i][j] + b[i][j];

}

printf("\nSum of two matrices: \n");

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

for (j = 0; j < c; ++j) {

printf("%d ", sum[i][j]);

if (j == c - 1) {

printf("\n\n");

}

}

return 0;

}

21) **Write a C Program to input two matrices of 5×5 multiply them and output the resultant matrix.**

**#include <stdio.h>**

**#define SIZE 5**

**int main()**

**{**

**int A[SIZE][SIZE];**

**int B[SIZE][SIZE];**

**int C[SIZE][SIZE];**

**int row, col, i, sum;**

**printf("Enter elements in matrix A of size %d x %d: \n", SIZE, SIZE);**

**for(row=0; row<SIZE; row++)**

**{**

**for(col=0; col<SIZE; col++)**

**{**

**scanf("%d", &A[row][col]);**

**}**

**}**

**printf("\n Enter elements in matrix B of size %d x %d: \n", SIZE, SIZE);**

**for(row=0; row<SIZE; row++)**

**{**

**for(col=0; col<SIZE; col++)**

**{**

**scanf("%d", &B[row][col]);**

**}**

**}**

**for(row=0; row<SIZE; row++)**

**{**

**for(col=0; col<SIZE; col++)**

**{**

**sum = 0;**

**for(i=0; i<SIZE; i++)**

**{**

**sum += A[row][i] \* B[i][col];**

**}**

**C[row][col] = sum;**

**}**

**}**

**printf("\n Product of matrix A \* B = \n");**

**for(row=0; row<SIZE; row++)**

**{**

**for(col=0; col<SIZE; col++)**

**{**

**printf("%d ", C[row][col]);**

**}**

**printf("\n");**

**}**

**return 0;**

**}**

**22)Write a C Program to find the sum of natural numbers using function.**

#include <stdio.h>

int sumOfNaturalNumbers(int first, int last);

int main()

{

int first, last, total;

printf("Enter first limit: ");

scanf("%d", &first);

printf("Enter last limit: ");

scanf("%d", &last);

sum = sumOfNaturalNumbers(first, last);

printf("Sum of natural numbers from %d to %d = %d", first, last, total);

return 0;

}

int sumOfNaturalNumbers(int first, int last)

{

if(first == last)

return first;

else

return first + sumOfNaturalNumbers(first + 1, last); }

**23) Write a C Program to find factorial of number using recursion.**

**#include<stdio.h>**

**int fact(int num);**

**int main() {**

**int num;**

**printf("Enter a integer: ");**

**scanf("%d",&num);**

**printf("Factorial of %d = %d", num, fact(num));**

**return 0;**

**}**

**int fact(int num) {**

**if (num>=1)**

**return num\*fact(num-1);**

**else**

**return 1;**

**}**

**24) . Write a C Program to generate the Fibonacci series.**

#include<stdio.h>

int fibo(int);

int main(void)

{

int a;

printf("Enter the value of a: ");

scanf("%d", &a);

for(int num = 0; num < a; num++)

{

printf("%d ", fibo(num));

}

return 0;

}

int fibo(int num)

{

if(num == 0 || num == 1)

{

return num;

}

else

{

return fibo(num-1) + fibo(num-2);

}

}

**25) Write a C Program using structure for entering details of the five students as**

**name, admission number, Date of birth, department and display all the details.**

**#include<stdio.h>**

**int main()**

**{**

**struct student**

**{**

**int roll\_num; char name[86]; int fee;**

**char DOB[105];**

**};**

**struct student stu[90]; int a,b;**

**printf("\n Enter number of students");**

**scanf("%d",&a); for(b=0;b<a;b++)**

**{**

**printf("\nenter the roll number");**

**scanf("%d",&stu[b].roll\_num);**

**printf("\n ENTER THE NAME");**

**scanf("%s",&stu[b].name);**

**printf("\n ENTER THE FEE");**

**scanf("%d",&stu[b].fee);**

**printf("\n ENTER THE DOB");**

**scanf("%s",&stu[b].DOB);**

**}**

**for(b=0;b<a;b++)**

**{**

**printf("\n Details of the student are %d",b+1);**

**printf("\n ROLL NO = %d",stu[b].roll\_num);**

**printf("\n NAME = %s",stu[b].name);**

**printf("\n FEE = %d",stu[b].fee);**

**printf("\n DOB = %s",stu[b].DOB);**

**}}**

**26) Write a C program to find length of string using pointers.**

#include<stdio.h>

int str\_lne(char\*);

void main()

{

char str[20]; int size;

printf("\n enter string : ");

gets(str);

size = str\_len(str);

printf("string length %s is : %d", str, size);

}

int str\_len(char\*a)

{

int total = 0;

while (\*a != '\0')

{

total++;

a++;

}

return total;

}

**27) Write a C program to copy one string to another using pointers.**

**#include<stdio.h>**

**int main()**

**{**

**char str[90],copy\_str[80];**

**char\*pstr,\*pcopy\_str;**

**pstr=str;**

**pcopy\_str=copy\_str;**

**printf("\n enter the string");**

**gets(str);**

**while(\*pstr!='\0')**

**{**

**\*pcopy\_str=\*pstr;**

**pstr++,pcopy\_str++;**

**}**

**\*pcopy\_str='\0';**

**printf("\n copied string is:");**

**pcopy\_str= copy\_str;**

**while(\*pcopy\_str!='\0')**

**{**

**printf("%c",\*pcopy\_str);**

**pcopy\_str++;**

**}**

**}**

**28) Write a C program to compare two strings using pointers.**

#include<stdio.h>

int main()

{

char string1[50],string2[60],\*a,\*b; int i, equal = 0;

printf("enter the first string: ");

scanf("%s",string1);

printf("enter the second string: ");

scanf("%s",string2);

a = string1;

b = string2;

while(\*a == \*b)

{

if ( \*a == '\0' || \*b == '\0' )

break;

a++;

b++;

}

if( \*a == '\0' && \*b == '\0' )

printf("\n\n entered strings are equal.");

else

printf("\n\n entered string are not equal");

}

**29) Write a C program to find the reverse of a string recursively and non-recursively.**

A)

**#include <stdio.h>**

**#include <string.h>**

**void reverse\_str(char\*, int, int);**

**int main()**

**{**

**char str\_arr[150]; printf("ENTER THE STRING:");**

**scanf("%s", &str\_arr);**

**reverse\_str(str\_arr, 0, strlen(str\_arr)-1);**

**printf("\nthe reversed string is: %s",str\_arr); return 0;**

**}**

**void reverse\_str(char \*a, int start, int b)**

**{**

**char ch;**

**if (start >= b)**

**return;**

**ch = \*(a+start);**

**\*(a+start) = \*(a+b);**

**\*(a+b) = ch;**

**reverse\_str(a, ++start, --b);**

**}**

**B)**

**#include <stdio.h>**

**#include <string.h>**

**int main()**

**{**

**char str[90],temp;**

**int a=0,b=0;**

**printf("\nEnter the string:");**

**gets(str);**

**b=strlen(str)-1;**

**while(a<b)**

**{**

**temp = str[b];**

**str[b]=str[a];**

**str[a]=temp;**

**a++;**

**b--;**

**}**

**printf("\n reversed string is: ");**

**puts(str);**

**}**

**30) Create a binary tree and output the data with 3 tree traversals**

#include <stdio.h>

#include <stdlib.h>

struct node

{

int info;

struct node\* left;

struct node\* right;

};

struct node\* newNode(int info)

{

struct node\* node = (struct node\*) malloc(sizeof(struct node));

node->info = info;

node->left = NULL;

node->right = NULL;

return(node);

}

void printPostorder(struct node\* node)

{

if (node == NULL)

return;

printPostorder(node->left);

printPostorder(node->right);

printf("%d ", node->info);

}

void printInorder(struct node\* node)

{

if (node == NULL) return;

printInorder(node->left);

printf("%d ", node->info);

printInorder(node->right);

}

void printPreorder(struct node\* node)

{

if (node == NULL) return;

printf("%d ", node->info);

printPreorder(node->left);

printPreorder(node->right);

}

int main()

{

struct node \*root = newNode(75);

root->left = newNode(126);

root->right = newNode(145);

root->left->left= newNode(63);

root->left->right= newNode(113);

printf("\n Pre-order transversal of binary tree is \n");

printPreorder(root);

printf("\n In-order transversal of binary tree is \n");

printInorder(root);

printf("\n Post-order transversal of binary tree is \n");

printPostorder(root);

getchar();

return 0;

}

**31)**

**# include <stdio.h>**

**# include <malloc.h>**

**#include <stdlib.h>**

**struct node**

**{**

**int info;**

**struct node \*left;**

**struct node \*right;**

**}\*root;**

**void find(int item,struct node \*\*par,struct node \*\*loc)**

**{**

**struct node \*ptr,\*ptrsave;**

**if(root==NULL)**

**{**

**\*loc=NULL;**

**\*par=NULL;**

**return;**

**}**

**if(item==root->info)**

**{**

**\*loc=root;**

**\*par=NULL;**

**return;**

**}**

**if(item<root->info)**

**ptr=root->left;**

**else**

**ptr=root->right;**

**ptrsave=root;**

**while(ptr!=NULL)**

**{**

**if(item==ptr->info)**

**{ \*loc=ptr;**

**\*par=ptrsave;**

**return;**

**}**

**ptrsave=ptr;**

**if(item<ptr->info)**

**ptr=ptr->left;**

**else**

**ptr=ptr->right;**

**}**

**\*loc=NULL;**

**\*par=ptrsave;**

**}**

**void insert(int item)**

**{ struct node \*tmp,\*parent,\*location;**

**find(item,&parent,&location);**

**if(location!=NULL)**

**{**

**printf("Item already present");**

**return;**

**}**

**tmp=(struct node \*)malloc(sizeof(struct node));**

**tmp->info=item;**

**tmp->left=NULL;**

**tmp->right=NULL;**

**if(parent==NULL)**

**root=tmp;**

**else**

**if(item<parent->info)**

**parent->left=tmp;**

**else**

**parent->right=tmp;**

**}**

**void case\_a(struct node \*par,struct node \*loc )**

**{**

**if(par==NULL)**

**root=NULL;**

**else**

**if(loc==par->left)**

**par->left=NULL;**

**else**

**par->right=NULL;**

**}**

**void case\_b(struct node \*par,struct node \*loc)**

**{**

**struct node \*child;**

**if(loc->left!=NULL)**

**child=loc->left;**

**else**

**child=loc->right;**

**if(par==NULL )**

**root=child;**

**else**

**if( loc==par->left)**

**par->left=child;**

**else**

**par->right=child;**

**}**

**void case\_c(struct node \*par,struct node \*loc)**

**{**

**struct node \*ptr,\*ptrsave,\*suc,\*parsuc;**

**ptrsave=loc;**

**ptr=loc->right;**

**while(ptr->left!=NULL)**

**{**

**ptrsave=ptr;**

**ptr=ptr->left;**

**}**

**suc=ptr;**

**parsuc=ptrsave;**

**if(suc->left==NULL && suc->right==NULL)**

**case\_a(parsuc,suc);**

**else**

**case\_b(parsuc,suc);**

**if(par==NULL)**

**root=suc;**

**else**

**if(loc==par->left)**

**par->left=suc;**

**else**

**par->right=suc;**

**suc->left=loc->left;**

**suc->right=loc->right;**

**}**

**int del(int item)**

**{**

**struct node \*parent,\*location;**

**if(root==NULL)**

**{**

**printf("Tree empty");**

**return 0;**

**}**

**find(item,&parent,&location);**

**if(location==NULL)**

**{**

**printf("Item not present in tree");**

**return 0;**

**}**

**if(location->left==NULL && location->right==NULL)**

**case\_a(parent,location);**

**if(location->left!=NULL && location->right==NULL)**

**case\_b(parent,location);**

**if(location->left==NULL && location->right!=NULL)**

**case\_b(parent,location);**

**if(location->left!=NULL && location->right!=NULL)**

**case\_c(parent,location);**

**free(location);**

**}**

**void display(struct node \*ptr,int level)**

**{**

**int i;**

**if ( ptr!=NULL )**

**{**

**display(ptr->right, level+1);**

**printf("\n");**

**for (i = 0; i < level; i++)**

**printf(" ");**

**printf("%d", ptr->info);**

**display(ptr->left, level+1);**

**}**

**}**

**int main()**

**{**

**int choice,num;**

**root=NULL;**

**while(1)**

**{**

**printf("\n");**

**printf("1.Insert\n");**

**printf("2.Delete\n");**

**printf("3.Display\n");**

**printf("4.Quit\n");**

**printf("Enter your choice : ");**

**scanf("%d",&choice);**

**switch(choice)**

**{**

**case 1:**

**printf("Enter the number to be inserted : ");**

**scanf("%d",&num);**

**insert(num);**

**break;**

**case 2:**

**printf("Enter the number to be deleted : ");**

**scanf("%d",&num);**

**del(num);**

**break;**

**case 3:**

**display(root,1);**

**break;**

**case 4:**

**exit(0);**

**default:**

**printf("Wrong choice\n");**

**}**

**}**

**return 0;}**

**32) Write a program to implement a single source shortest path algorithm. Either Bellman-Ford or Dijkstra’s algorithm.**

#include <limits.h>

#include <stdio.h>

#define V 9

int minDistance(int dist[], bool sptSet[])

{

int min = INT\_MAX, min\_index;

for (int v = 0; v < V; v++)

if (sptSet[v] == false && dist[v] <= min)

min = dist[v], min\_index = v;

return min\_index;

}

void printSolution(int dist[])

{

printf("Vertex \t\t Distance from Source\n");

for (int i = 0; i < V; i++)

printf("%d \t\t %d\n", i, dist[i]);

}

void dijkstra(int graph[V][V], int src)

{

int dist[V];

bool sptSet[V];

for (int i = 0; i < V; i++)

dist[i] = INT\_MAX, sptSet[i] = false;

dist[src] = 0;

for (int count = 0; count < V - 1; count++) {

int u = minDistance(dist, sptSet);

sptSet[u] = true;

for (int v = 0; v < V; v++)

if (!sptSet[v] && graph[u][v] && dist[u] != INT\_MAX

&& dist[u] + graph[u][v] < dist[v])

dist[v] = dist[u] + graph[u][v];

}

printSolution(dist);

}

int main()

{

int graph[V][V] = { { 0, 4, 0, 0, 0, 0, 0, 8, 0 },

{ 4, 0, 8, 0, 0, 0, 0, 11, 0 },

{ 0, 8, 0, 7, 0, 4, 0, 0, 2 },

{ 0, 0, 7, 0, 9, 14, 0, 0, 0 },

{ 0, 0, 0, 9, 0, 10, 0, 0, 0 },

{ 0, 0, 4, 14, 10, 0, 2, 0, 0 },

{ 0, 0, 0, 0, 0, 2, 0, 1, 6 },

{ 8, 11, 0, 0, 0, 0, 1, 0, 7 },

{ 0, 0, 2, 0, 0, 0, 6, 7, 0 } };

dijkstra(graph, 0);

return 0;

}

**33)** **Write a program to find All-to-all Shortest paths in a Graph.**

#include <bits/stdc++.h>

using namespace std;

void add\_edge(vector<int> adj[], int src, int dest)

{

adj[src].push\_back(dest);

adj[dest].push\_back(src);

}

bool BFS(vector<int> adj[], int src, int dest, int v, int pred[], int dist[])

{

list<int> queue;

bool visited[v];

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

visited[i] = false;

dist[i] = INT\_MAX;

pred[i] = -1;

}

visited[src] = true;

dist[src] = 0;

queue.push\_back(src);

while (!queue.empty()) {

int u = queue.front();

queue.pop\_front();

for (int i = 0; i < adj[u].size(); i++) {

if (visited[adj[u][i]] == false) {

visited[adj[u][i]] = true;

dist[adj[u][i]] = dist[u] + 1;

pred[adj[u][i]] = u;

queue.push\_back(adj[u][i]);

if (adj[u][i] == dest)

return true;

}

}

}

return false;

}

void printShortestDistance(vector<int> adj[], int s,

int dest, int v)

{

int pred[v], dist[v];

if (BFS(adj, s, dest, v, pred, dist) == false) {

cout << "Given source and destination"

<< " are not connected";

return;

}

vector<int> path;

int crawl = dest;

path.push\_back(crawl);

while (pred[crawl] != -1) {

path.push\_back(pred[crawl]);

crawl = pred[crawl];

}

cout << "Shortest path length is : "

<< dist[dest];

cout << "\n Path is::\n";

for (int i = path.size() - 1; i >= 0; i--)

cout << path[i] << " ";

}

int main()

{

int v = 8;

vector<int> adj[v];

add\_edge(adj, 0, 1);

add\_edge(adj, 0, 3);

add\_edge(adj, 1, 2);

add\_edge(adj, 3, 4);

add\_edge(adj, 3, 7);

add\_edge(adj, 4, 5);

add\_edge(adj, 4, 6);

add\_edge(adj, 4, 7);

add\_edge(adj, 5, 6);

add\_edge(adj, 6, 7);

int source = 0, dest = 7;

printShortestDistance(adj, source, dest, v);

return 0;

}

**34)Write a C program to implement the STACK operation using array as a data**

**structure. Users must be given the following choices to perform relevant tasks.**

**a. Push an element on to the STACK.**

**b. Pop and element from the STACK.**

**c. Peek the STACK.**

**d. Display the STACK.**

**e. Exit the program.**

#include<stdio.h>

#define MAX 50

int stack[MAX],choice , n , top ,x ,i;

void push(void);

void pop(void);

void display(void);

void peek(void);

int main()

{

top=-1;

printf("\n enter the size of stack:");

scanf("%d", &n);

printf("\n\t stack operations used in this array");

printf("\n\t ");

printf("\n\t 1.PUSH\n\t 2.POP\n\t 3.DISPLAY\n\t 4.PEEK\n\t 5.EXIT");

do{

printf("\n enter the choice:");

scanf("%d", &choice); switch(choice)

{

case 1:

{

push();

break;

}

case 2:

{

pop();

break;

}

case 3:

{

display();

break;

}

case 4:

{

peek();

break;

}

case 5:

{

printf("\n\t exit ");

break;

}

default:

{

printf ("\n\t entered number is wrong");

}}}

while(choice!=5);

return 0;

}

void push()

{

if(top>=n-1)

{

printf("\n\t stack overflow");

}

else

{

printf("enter a number to be pushed:");

scanf("%d", &x); top++; stack[top]=x;

}

}

void pop()

{

if(top<=-1)

{

printf("\n\t stack is under flow");

}

else

{

printf("\n\t element which popped is %d", stack[top]);

top--;

}

}

void display()

{

if(top>=0)

{

printf("\n THE ELEMENTS IN STACK \n");

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

printf("\n %d", stack[i]);

printf("\n next choice");

}

else

{

printf("\n empty stack");

}

}

void peek()

{

printf("\n peek element is %d", stack[top]);

}

**35) Write a C program to reverse a string using STACK.**

**#include <stdio.h>**

**#include <string.h>**

**#define max 100**

**int top, stack[max];**

**void push(char x)**

**{**

**if(top == max-1)**

**{**

**printf("stack is overflow");**

**}**

**else**

**{**

**stack[++top]=x;**

**}**

**}**

**void pop()**

**{**

**printf("%c", stack[top--]);**

**}**

**main()**

**{**

**char str[50];**

**printf("string is : \n");**

**scanf("%s", &str);**

**int len = strlen(str);**

**int i;**

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

**push(str[i]);**

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

**pop();**

**}**

**36) Write a C program to convert the given infix expression to post-fix expression using STACK.**

#include<stdio.h>

#include<string.h>

#include<ctype.h>

#define MAX 70

char st[MAX];

int top =-1;

void push(char st[],char);

char pop(char st[]);

void InfixtoPostfix(char source[],char target[]);

int getPriority(char);

int main()

{

char infix[100], postfix[100];

printf("\n ENTER THE EXPRESSION OF INFIX");

gets(infix); strcpy(postfix,""); InfixtoPostfix(infix, postfix);

printf("\n THE POSTFIX EXPRESSION IS ");

puts(postfix); return 0;

}

void InfixtoPostfix(char source[],char target[])

{

int i=0,j=0;

char temp;

strcpy(target,"");

while(source[i]!='\0')

{

if(source[i]=='(')

{

push(st, source[i]);

i++;

}

else if(source[i]==')')

{

while((top!=-1)&&(st[top]!='('))

{

target[j]=pop(st);

j++;

}

if(top==-1)

{

printf("\n incorrect expression");

exit(1);

}

temp= pop(st);

i++;

}

else if(isdigit(source[i])|| isalpha(source[i]))

{

target[j]=source[i];

j++;

i++;

}

else if(source[i]=='+'|| source[i]=='-'|| source[i]=='\*'|| source[i]== '/'|| source[i]=='%')

{

while((top!=-1)&&(st[top]!='(')&& (getPriority(st[top])>getPriority (source[i])))

{

target[j]=pop(st);

j++;

}

push(st, source[i]);

i++;

}

else

{

printf("\n incorrect element in expression");

exit(1);

}

}

while((top!=-1)&&(st[top]!='('))

{

target[j]=pop(st);

j++;

}

target[j]='\0';

}

int getPriority (char op)

{

if(op=='/'||op=='\*'||op=='%')

return 1;

else if(op=='+'||op=='-')

return 0;

}

void push(char st[],char val)

{

if(top==MAX -1)

printf("\n stack is overflow");

else

{

top++;

st[top]=val;

}

}

char pop(char st[])

{

char val=' ';

if(top==-1)

printf("\n stack is underflow");

else

{

val=st[top];

top--;

}

return val;

}

**37) Write a C program to convert the given in-fix expression to pre-fix expression using STACK.**

**#include<stdio.h>**

**#include<string.h>**

**#include<ctype.h>**

**#define MAX 50**

**char st[MAX];**

**int top=-1;**

**void reverse(char str[]);**

**void push(char st[],char);**

**char pop(char st[]);**

**void Infixtopostfix(char source[],char target[]);**

**int getPriority(char);**

**char infix[100],postfix[100],temp[100];**

**int main()**

**{**

**printf("\n enter infix expression");**

**gets(infix);**

**reverse(infix);**

**strcpy(postfix,"");**

**Infixtopostfix(temp, postfix);**

**printf("\n the corresponding postfix expression");**

**puts(postfix);**

**strcpy(temp,"");**

**reverse(postfix);**

**printf("\n prefix expression is");**

**puts(temp);**

**return 0;**

**}**

**void reverse(char str[])**

**{**

**int len,i=0,j=0;;**

**len=strlen(str); j=len-1;**

**while(j>=0)**

**{**

**if(str[j]=='(')**

**temp[i]=')';**

**else if(str[j]==')')**

**temp[i]='(';**

**else temp[i]=str[j];**

**i++;**

**j--;**

**}**

**temp[i]='\0';**

**}**

**void Infixtopostfix(char source[], char target[])**

**{**

**int i=0,j=0;**

**char temp;**

**strcpy(target,"");**

**while(source[i]!='\0')**

**{**

**if(source[i]=='(')**

**{**

**push(st, source[i]);**

**i++;**

**}**

**else if(source[i]==')')**

**{**

**while((top!=-1)&&(st[top]!='('))**

**{**

**target[j]=pop(st);**

**j++;**

**}**

**if(top==-1)**

**{**

**printf("\n wrong expression");**

**exit(1);**

**}**

**temp=pop(st);**

**i++;**

**}**

**else if(isdigit(source[i])||isalpha(source[i]))**

**{**

**target[j]= source[i];**

**j++;**

**i++;**

**}**

**else if(source[i]=='+'||source[i]=='-'||source[i]=='\*'||source[i]=='/'||source[i]=='%')**

**{**

**while((top!=-1)&&(st[top]!='(') &&(getPriority(st[top])> getPriority(source[i])))**

**{**

**target[j]= pop(st);**

**j++;**

**}**

**push(st, source[i]);**

**i++;**

**}**

**else**

**{**

**printf("\n incorrect elements in expression");**

**exit(1);**

**}**

**}**

**while((top!=-1)&&(st[top]!='('))**

**{**

**target[j]= pop(st);**

**j++;**

**}**

**target[j]='\0';**

**}**

**int getPriority(char op)**

**{**

**if(op=='/'||op=='\*'||op=='%') return 1;**

**else if(op=='+'||op=='-')**

**return 0;**

**}**

**void push(char st[], char val)**

**{**

**if(top==MAX -1)**

**printf("\n stack is overflow");**

**else**

**{**

**top++;**

**st[top]=val;**

**}**

**}**

**char pop(char st[])**

**{**

**char val= ' ';**

**if(top==-1)**

**printf("\n stack is underflow");**

**else**

**{**

**val=st[top];**

**top--;**

**}**

**return val;**

**}**

**38)Write a C program to evaluate the given pre-fix expression and post-fix**

**expressions.**

#include<stdio.h>

int stack[50];

int top = -1;

void push(int a)

{

stack[++top] = a;

}

int pop()

{

return stack[top--];

}

int main()

{

char exp[50]; char \*e;

int num1, num2, num3, num;

printf("enter expression : ");

scanf("%s" , exp); e = exp;

while(\*e != '\0')

{

if(isdigit(\*e))

{

num = \*e - 48;

push(num);

}

else

{

num1 = pop();

num2 = pop();

switch(\*e)

{

case '+':

{

num3 = num1 + num2;

break;

}

case '-':

{

num3 = num2 - num1;

break;

}

case '\*':

{

num3 = num1 \* num2;

break;

}

case '/':

{

num3 = num2 / num1;

break;

}

}

push(num3);

}

e++;

}

printf("\n expression result is %s = %d\n\n", exp, pop());

return 0;

}

**39)Write a C program to implement a Linear-Queue, user must choose the following options:**

**a. Add an element to the Queue – EnQueue.**

**b. Remove an element from the Queue – DeQueue.**

**c. Display the elements of the Queue.**

**d. Terminate the program.**

**#include<stdio.h>**

**#define MAX 50**

**int queue[MAX];**

**int front=-1,rear=-1;**

**void insert(void);**

**int delete\_element(void);**

**int peep(void);**

**void display(void);**

**int main()**

**{**

**int option, val;**

**do{**

**printf("\n\n\*\*\*\*\*MAIN MENU\*\*\*\*\*");**

**printf("\n 1. ENQUEUE");**

**printf("\n 2. DEQUEUE");**

**printf("\n 3. PEEK");**

**printf("\n 4. DISPLAY THE QUEUE");**

**printf("\n 5. EXIT");**

**printf("\n \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");**

**printf("\n\n PRESS YOUR OPTION");**

**scanf("%d", &option);**

**switch(option)**

**{**

**case 1:**

**insert();**

**break;**

**case 2:**

**val=delete\_element();**

**if(val!=-1)**

**printf("\n Deleted number is %d", val);**

**break;**

**case 3:**

**val= peep();**

**if(val!=-1)**

**printf("\n first value in the queue is %d", val);**

**break;**

**case 4:**

**display();**

**break;**

**}**

**}while(option!=5);**

**return 0;**

**}**

**void insert()**

**{**

**int num;**

**printf("\n Enter the number to enqueue");**

**scanf("%d", &num);**

**if(rear==MAX-1)**

**printf("\n OVER-FLOW");**

**else if(front==-1&&rear==-1)**

**front=rear=0;**

**else**

**rear++;**

**queue[rear]=num;**

**}**

**int delete\_element()**

**{**

**int val;**

**if(front==-1 || front>rear)**

**{**

**printf("\n underflow");**

**return -1;**

**}**

**else**

**{**

**val=queue[front]; front++;**

**if(front>rear)**

**front=rear=-1;**

**return val;**

**}**

**}**

**int peep()**

**{**

**if(front==-1 || front> rear)**

**{**

**printf("\n empty queue");**

**return -1;**

**}**

**else**

**{**

**return queue[front];**

**}**

**}**

**void display()**

**{**

**int i; printf("\n");**

**if(front==-1 || front > rear)**

**printf("\n empty queue");**

**else**

**{**

**for(i=front;i<=rear;i++)**

**printf("\t %d", queue[i]);**

**}**

**}**

**40)Write a C program to implement a Circular-Queue, user must choose the**

**following options:**

**a. Add an element to the Queue – EnQueue.**

**b. Remove an element from the Queue – DeQueue.**

**c. Display the elements of the Queue.**

**d. Terminate the program.**

#include<stdio.h>

#define MAX 50

void insertq(int[], int);

void deleteq(int[]);

void display(int[]);

int front = - 1;

int rear = - 1;

int main()

{

int n, ch;

int queue[MAX];

do{

printf("\n\n CIRCULAR QUEUE CHOICES:\n1. ENQUEUE \n2. DEQUEUE\n3. DISPLAY\n0. EXIT"); printf("\n PRESS THE CHOICE: ");

scanf("%d", &ch);

switch (ch)

{

case 1:

printf("\n enter number: ");

scanf("%d", &n); insertq(queue, n);

break;

case 2:

deleteq(queue);

break;

case 3:

display(queue);

break;

}8 MAX - 1 && front > 0)

{

rear = 0;

}

else

{

rear++;

}

queue[rear] = item;

}

void display(int queue[])

{

int i;

printf("\n");

if (front > rear)

{

for (i = front; i < MAX; i++)

{

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

}

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

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

}

else

{

for (i = front; i <= rear; i++)

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

}

}

void deleteq(int queue[])

{

if (front == - 1)

{

printf("queue is underflow ");

}

else if (front == rear)

{

printf("\n %d removed", queue[front]);

front = - 1;

rear = - 1;

}

else

{

printf("\n %d REMOVED", queue[front]);

front++;

}

}

41) **Write a C program to create a single linked list with 5 nodes. (5 integers are taken from user input) and display the linked-list elements.**

#include <stdio.h>

#include <stdlib.h>

struct node

{

int num;

struct node \*nextptr;

}\*snode;

void createNodeList(int n);

void displayList();

int main()

{

printf("\n\n Creation and display of Singly Linked List :\n");

int n;

printf(" Input the number of nodes : ");

scanf("%d", &n);

createNodeList(n);

printf("\n Data entered in the list : \n");

displayList();

return 0;

}

void createNodeList(int n)

{

struct node \*fnNode, \*tmp;

int num, i;

snode = (struct node \*)malloc(sizeof(struct node));

if(snode == NULL)

{

printf(" Memory can not be allocated.");

}

else

{

printf(" Input data for node 1 : ");

scanf("%d", &num);

snode->num = num;

snode->nextptr = NULL;

tmp = snode;

for(i=2; i<=n; i++)

{

fnNode = (struct node \*)malloc(sizeof(struct node));

if(fnNode == NULL)

{

printf(" Memory can not be allocated.");

break;

}

else

{

printf(" Input data for node %d : ", i);

scanf(" %d", &num);

fnNode->num = num;

fnNode->nextptr = NULL;

tmp->nextptr = fnNode;

tmp = tmp->nextptr;

}

}

}

}

void displayList()

{

struct node \*tmp;

if(snode == NULL)

{

printf(" List is empty.");

}

else

{

tmp = snode;

while(tmp != NULL)

{

printf(" Data = %d\n", tmp->num);

tmp = tmp->nextptr;

}

}

}

42) **. Write a C program to search an element in a singly-linked list.**

#include <stdio.h>

#include <stdlib.h>

struct node

{

int num;

struct node \*nextptr;

}

stnode, \*enode;

int SearchElement(int);

void main()

{

int n,i,FindElem,FindPlc;

stnode.nextptr=NULL;

enode=&stnode;

printf(" Input the number of nodes : ");

scanf("%d", &n);

printf("\n");

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

{

enode->nextptr=(struct node \*)malloc(sizeof(struct node));

printf(" Input data for node %d : ",i+1);

scanf("%d",&enode->num);

enode=enode->nextptr;

}

enode->nextptr=NULL;

printf("\n Data entered in the list are :\n");

enode=&stnode;

while(enode->nextptr!=NULL)

{

printf(" Data = %d\n",enode->num);

enode=enode->nextptr;

}

printf("\n");

printf(" Input the element to be searched : ");

scanf("%d",&FindElem);

FindPlc=SearchElement(FindElem);

if(FindPlc<=n)

printf(" Element found at node %d \n\n",FindPlc);

else

printf(" This element does not exists in linked list.\n\n");

}

int SearchElement(int FindElem)

{

int ctr=1;

enode=&stnode;

while(enode->nextptr!=NULL)

{

if(enode->num==FindElem)

break;

else

ctr++;

enode=enode->nextptr;

}

return ctr;

}

**43. Write a C program to perform the following tasks:**

**a. Insert a node at the beginning of a singly-linked list.**

**b. Insert a node at end of a singly-linked list.**

**c. Insert a node at the middle of a singly-linked list.**

**d. Delete a node from the beginning of the singly-linked list.**

**e. Delete a node from the end of a singly-linked list**

A)

#include <stdio.h>

#include <stdlib.h>

struct node

{

int num;

struct node \*nextptr;

}\*stnode;

void createNodeList(int n);

void NodeInsertatBegin(int num);

void displayList();

int main()

{

int n,num;

printf(" Input the number of nodes : ");

scanf("%d", &n);

createNodeList(n);

printf("\n Data entered in the list are : \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();

return 0;

}

void createNodeList(int n)

{

struct node \*fnNode, \*tmp;

int num, i;

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

if(stnode == NULL)

{

printf(" Memory can not be allocated.");

}

else

{

printf(" Input data for node 1 : ");

scanf("%d", &num);

stnode-> num = num;

stnode-> nextptr = NULL;

tmp = stnode;

for(i=2; i<=n; i++)

{

fnNode = (struct node \*)malloc(sizeof(struct node));

if(fnNode == NULL)

{

printf(" Memory can not be allocated.");

break;

}

else

{

printf(" Input data for node %d : ", i);

scanf(" %d", &num);

fnNode->num = num;

fnNode->nextptr = NULL;

tmp->nextptr = fnNode;

tmp = tmp->nextptr;

}

}

}

}

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;

fnNode->nextptr = stnode;

stnode = fnNode;

}

}

void displayList()

{

struct node \*tmp;

if(stnode == NULL)

{

printf(" No data found in the list.");

}

else

{

tmp = stnode;

while(tmp != NULL)

{

printf(" Data = %d\n", tmp->num);

tmp = tmp->nextptr;

}

}

}

B)

#include <stdio.h>

#include <stdlib.h>

struct node

{

int num;

struct node \*nextptr;

}\*stnode;

void createNodeList(int n);

void NodeInsertatEnd(int num);

void displayList();

int main()

{

int n,num;

printf(" Input the number of nodes : ");

scanf("%d", &n);

createNodeList(n);

printf("\n Data entered in the list are : \n");

displayList();

printf("\n Input data to insert at the end of the list : ");

scanf("%d", &num);

NodeInsertatEnd(num);

printf("\n Data, after inserted in the list are : \n");

displayList();

return 0;

}

void createNodeList(int n)

{

struct node \*fnNode, \*tmp;

int num, i;

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

if(stnode == NULL)

{

printf(" Memory can not be allocated.");

}

else

{

printf(" Input data for node 1 : ");

scanf("%d", &num);

stnode-> num = num;

stnode-> nextptr = NULL;

tmp = stnode;

for(i=2; i<=n; i++)

{

fnNode = (struct node \*)malloc(sizeof(struct node));

if(fnNode == NULL)

{

printf(" Memory can not be allocated.");

break;

}

else

{

printf(" Input data for node %d : ", i);

scanf(" %d", &num);

fnNode->num = num;

fnNode->nextptr = NULL;

tmp->nextptr = fnNode;

tmp = tmp->nextptr;

}

}

}

}

void NodeInsertatEnd(int num)

{

struct node \*fnNode, \*tmp;

fnNode = (struct node\*)malloc(sizeof(struct node));

if(fnNode == NULL)

{

printf(" Memory can not be allocated.");

}

else

{

fnNode->num = num;

fnNode->nextptr = NULL;

tmp = stnode;

while(tmp->nextptr != NULL)

tmp = tmp->nextptr;

tmp->nextptr = fnNode;

}

}

void displayList()

{

struct node \*tmp;

if(stnode == NULL)

{

printf(" No data found in the empty list.");

}

else

{

tmp = stnode;

while(tmp != NULL)

{

printf(" Data = %d\n", tmp->num);

tmp = tmp->nextptr;

}

}

}

C)

#include <stdio.h>

#include <stdlib.h>

struct node

{

int num; //Data of the node

struct node \*nextptr; //Address of the node

}\*stnode;

void createNodeList(int n); //function to create the list

void insertNodeAtMiddle(int num, int pos); //function to insert node at the middle

void displayList(); //function to display the list

int main()

{

int n,num,pos;

printf(" Input the number of nodes: ");

scanf("%d", &n);

createNodeList(n);

printf("\n Data entered in the list are : \n");

displayList();

printf("\n Input data to insert in the middle of the list : ");

scanf("%d", &num);

printf(" Input the position to insert new node : " );

scanf("%d", &pos);

if(pos<=1 || pos>=n)

{

printf("\n Insertion can not be possible in that position.\n ");

}

if(pos>1 && pos<n)

{

insertNodeAtMiddle(num, pos);

printf("\n Insertion completed successfully.\n ");

}

printf("\n The new list are : \n");

displayList();

return 0;

}

void createNodeList(int n)

{

struct node \*fnNode, \*tmp;

int num, i;

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

if(stnode == NULL) //check whether the stnode is NULL and if so no memory allocation

{

printf(" Memory can not be allocated.");

}

else

{

// reads data for the node through keyboard

printf(" Input data for node 1 : ");

scanf("%d", &num);

stnode-> num = num;

stnode-> nextptr = NULL; //Links the address field to NULL

tmp = stnode;

//Creates n nodes and adds to linked list

for(i=2; i<=n; i++)

{

fnNode = (struct node \*)malloc(sizeof(struct node));

if(fnNode == NULL) //check whether the fnnode is NULL and if so no memory allocation

{

printf(" Memory can not be allocated.");

break;

}

else

{

printf(" Input data for node %d : ", i);

scanf(" %d", &num);

fnNode->num = num;

fnNode->nextptr = NULL;

tmp->nextptr = fnNode;

tmp = tmp->nextptr;

}

}

}

}

void insertNodeAtMiddle(int num, int pos)

{

int i;

struct node \*fnNode, \*tmp;

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 = NULL;

tmp = stnode;

for(i=2; i<=pos-1; i++)

{

tmp = tmp->nextptr;

if(tmp == NULL)

break;

}

if(tmp != NULL)

{

fnNode->nextptr = tmp->nextptr; //Links the address part of new node

tmp->nextptr = fnNode;

}

else

{

printf(" Insert is not possible to the given position.\n");

}

}

}

void displayList()

{

struct node \*tmp;

if(stnode == NULL)

{

printf(" No data found in the empty list.");

}

else

{

tmp = stnode;

while(tmp != NULL)

{

printf(" Data = %d\n", tmp->num); // prints the data of current node

tmp = tmp->nextptr;

}

}

}

D)

#include <stdio.h>

#include <stdlib.h>

struct node

{

int num; //Data of the node

struct node \*nextptr; //Address of the node

}\*stnode;

void createNodeList(int n); //function to create the list

void FirstNodeDeletion(); //function to delete the first node

void displayList(); //function to display the list

int main()

{

int n,num,pos;

printf(" Input the number of nodes : ");

scanf("%d", &n);

createNodeList(n);

printf("\n Data entered in the list are : \n");

displayList();

FirstNodeDeletion();

printf("\n Data, after deletion of first node : \n");

displayList();

return 0;

}

void createNodeList(int n)

{

struct node \*fnNode, \*tmp;

int num, i;

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

if(stnode == NULL) //check whether the stnode is NULL and if so no memory allocation

{

printf(" Memory can not be allocated.");

}

else

{

printf(" Input data for node 1 : ");

scanf("%d", &num);

stnode-> num = num;

stnode-> nextptr = NULL; //Links the address field to NULL

tmp = stnode;

for(i=2; i<=n; i++)

{

fnNode = (struct node \*)malloc(sizeof(struct node));

if(fnNode == NULL) //check whether the fnnode is NULL and if so no memory allocation

{

printf(" Memory can not be allocated.");

break;

}

else

{

printf(" Input data for node %d : ", i);

scanf(" %d", &num);

fnNode->num = num; // links the num field of fnNode with num

fnNode->nextptr = NULL;

tmp->nextptr = fnNode; // links previous node i.e. tmp to the fnNode

tmp = tmp->nextptr;

}

}

}

}

void FirstNodeDeletion()

{

struct node \*toDelptr;

if(stnode == NULL)

{

printf(" There are no node in the list.");

}

else

{

toDelptr = stnode;

stnode = stnode->nextptr;

printf("\n Data of node 1 which is being deleted is : %d\n", toDelptr->num);

free(toDelptr); // Clears the memory occupied by first node

}

}

void displayList()

{

struct node \*tmp;

if(stnode == NULL)

{

printf(" No data found in the list.");

}

else

{

tmp = stnode;

while(tmp != NULL)

{

printf(" Data = %d\n", tmp->num); // prints the data of current node

tmp = tmp->nextptr;

}

}

}

E)

#include <stdio.h>

#include <stdlib.h>

struct node

{

int num; //Data of the node

struct node \*nextptr; //Address of the node

}\*stnode;

void createNodeList(int n); //function to create the list

void LastNodeDeletion(); //function to delete the last nodes

void displayList(); //function to display the list

int main()

{

int n,num,pos;

printf(" Input the number of nodes : ");

scanf("%d", &n);

createNodeList(n);

printf("\n Data entered in the list are : \n");

displayList();

LastNodeDeletion();

printf("\n The new list after deletion the last node are : \n");

displayList();

return 0;

}

void createNodeList(int n)

{

struct node \*fnNode, \*tmp;

int num, i;

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

if(stnode == NULL) //check whether the stnode is NULL and if so no memory allocation

{

printf(" Memory can not be allocated.");

}

else

{

printf(" Input data for node 1 : ");

scanf("%d", &num);

stnode-> num = num;

stnode-> nextptr = NULL; //Links the address field to NULL

tmp = stnode;

//Creates n nodes and adds to linked list

for(i=2; i<=n; i++)

{

fnNode = (struct node \*)malloc(sizeof(struct node));

if(fnNode == NULL) //check whether the fnnode is NULL and if so no memory allocation

{

printf(" Memory can not be allocated.");

break;

}

else

{

printf(" Input data for node %d : ", i);

scanf(" %d", &num);

fnNode->num = num; // links the num field of fnNode with num

fnNode->nextptr = NULL; // links the address field of fnNode with NULL

tmp->nextptr = fnNode; // links previous node i.e. tmp to the fnNode

tmp = tmp->nextptr;

}

}

}

}

// Deletes the last node of the linked list

void LastNodeDeletion()

{

struct node \*toDelLast, \*preNode;

if(stnode == NULL)

{

printf(" There is no element in the list.");

}

else

{

toDelLast = stnode;

preNode = stnode;

while(toDelLast->nextptr != NULL)

{

preNode = toDelLast;

toDelLast = toDelLast->nextptr;

}

if(toDelLast == stnode)

{

stnode = NULL;

}

else

{

preNode->nextptr = NULL;

}

/\* Delete the last node \*/

free(toDelLast);

}

}

// function to display the entire list

void displayList()

{

struct node \*tmp;

if(stnode == NULL)

{

printf(" No data found in the empty list.");

}

else

{

tmp = stnode;

while(tmp != NULL)

{

printf(" Data = %d\n", tmp->num); // prints the data of current node

tmp = tmp->nextptr;

}

}

}

**44) Write a C program to create a doubly linked list with 5 nodes.**

#include <stdio.h>

#include <stdlib.h>

struct node {

int num;

struct node \* preptr;

struct node \* nextptr;

}\*stnode, \*ennode;

void DlListcreation(int n);

void displayDlList();

int main()

{

int n;

stnode = NULL;

ennode = NULL;

printf(" Input the number of nodes : ");

scanf("%d", &n);

DlListcreation(n);

displayDlList();

return 0;

}

void DlListcreation(int n)

{

int i, num;

struct node \*fnNode;

if(n >= 1)

{

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

if(stnode != NULL)

{

printf(" Input data for node 1 : "); // assigning data in the first node

scanf("%d", &num);

stnode->num = num;

stnode->preptr = NULL;

stnode->nextptr = NULL;

ennode = stnode;

// putting data for rest of the nodes

for(i=2; i<=n; i++)

{

fnNode = (struct node \*)malloc(sizeof(struct node));

if(fnNode != NULL)

{

printf(" Input data for node %d : ", i);

scanf("%d", &num);

fnNode->num = num;

fnNode->preptr = ennode; // new node is linking with the previous node

fnNode->nextptr = NULL;

ennode->nextptr = fnNode; // previous node is linking with the new node

ennode = fnNode; // assign new node as last node

}

else

{

printf(" Memory can not be allocated.");

break;

}

}

}

else

{

printf(" Memory can not be allocated.");

}

}

}

void displayDlList()

{

struct node \* tmp;

int n = 1;

if(stnode == NULL)

{

printf(" No data found in the List yet.");

}

else

{

tmp = stnode;

printf("\n\n Data entered on the list are :\n");

while(tmp != NULL)

{

printf(" node %d : %d\n", n, tmp->num);

n++;

tmp = tmp->nextptr; // current pointer moves to the next node

}

}

}

**45. Write a C program to create a circular linked list with 5 nodes.**

#include <stdio.h>

#include <stdlib.h>

struct node {

int num;

struct node \* nextptr;

}\*stnode;

void ClListcreation(int n);

void displayClList();

int main()

{

int n;

stnode = NULL;

printf("\n\n Circular Linked List : Create and display a circular linked list :\n");

printf("-----------------------------------------------------------------------\n");

printf(" Input the number of nodes : ");

scanf("%d", &n);

ClListcreation(n);

displayClList();

return 0;

}

void ClListcreation(int n)

{

int i, num;

struct node \*preptr, \*newnode;

if(n >= 1)

{

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

printf(" Input data for node 1 : ");

scanf("%d", &num);

stnode->num = num;

stnode->nextptr = NULL;

preptr = stnode;

for(i=2; i<=n; i++)

{

newnode = (struct node \*)malloc(sizeof(struct node));

printf(" Input data for node %d : ", i);

scanf("%d", &num);

newnode->num = num;

newnode->nextptr = NULL; // next address of new node set as NULL

preptr->nextptr = newnode; // previous node is linking with new node

preptr = newnode; // previous node is advanced

}

preptr->nextptr = stnode; //last node is linking with first node

}

}

void displayClList()

{

struct node \*tmp;

int n = 1;

if(stnode == NULL)

{

printf(" No data found in the List yet.");

}

else

{

tmp = stnode;

printf("\n\n Data entered in the list are :\n");

do {

printf(" Data %d = %d\n", n, tmp->num);

tmp = tmp->nextptr;

n++;

}while(tmp != stnode);

}

}

**46) Write a C program to implement the stack using linked list.**

#include<stdio.h>

#include<malloc.h>

typedef struct node

{

char s\_name[20],s\_address[50];

int s\_marks;

struct node \*next;

}s;

s \*push(s\*);

s \*pop(s \*);

void display(s \*);

int main()

{

s \*top=NULL;

int ch, x ,c=0;

printf("Enter 1 for push\n");

printf("Enter 2 for pop\n");

printf("Enter 3 for display\n");

do

{

printf("Enter your choice: ");

scanf("%d", &ch);

switch(ch)

{

case 1:

top=push(top);

break;

case 2:

top=pop(top);

break;

case 3:

display(top);

break;

}

printf("do you want to continue press 1: ");

scanf("%d", &c);

}while(c==1);

}

s \*push(s \*top)

{

s \*p;

p=(s \*)malloc(sizeof(s));

if(p==NULL)

{

printf("no memory allocated");

}

else

{

printf("\n Enter the student name: ");

scanf("%s", &p->s\_name);

printf("Enter student address: ");

scanf("%s", &p->s\_address);

printf("Enter the marks of students: ");

scanf("%d", &p->s\_marks);

p->next=top;

top=p;

}

return(top);

}

s \*pop(s \*top)

{

s \*p;

if(top==NULL)

{

printf("nothing to pop");

}

else

{

printf("\n The student name is: %s",top->s\_name);

printf("\n The student address is: %s",top->s\_address);

printf("\n The marks of the student is: %d",top->s\_marks);

top=top->next;

}

return(top);

}

void display(s \*top)

{

if(top==NULL)

{

printf("nothing to display");

}

else

{

while(top!=NULL)

{

printf("\n The student name is: %s",top->s\_name);

printf("\n The student address is: %s",top->s\_address);

printf("\n The marks of the student is: %d",top->s\_marks);

top=top->next;

}

}

}

**47) Write a C program to implement the queue using a linked list.**

#include <stdio.h>

#include <stdlib.h>

struct node

{

int info;

struct node \*ptr;

}\*front,\*rear,\*temp,\*front1;

int frontelement();

void enq(int data);

void deq();

void empty();

void display();

void create();

void queuesize();

int count = 0;

void main()

{

int no, ch, e;

printf("\n 1 - Enque");

printf("\n 2 - Deque");

printf("\n 3 - Front element");

printf("\n 4 - Empty");

printf("\n 5 - Exit");

printf("\n 6 - Display");

printf("\n 7 - Queue size");

create();

while (1)

{

printf("\n Enter choice : ");

scanf("%d", &ch);

switch (ch)

{

case 1:

printf("Enter data : ");

scanf("%d", &no);

enq(no);

break;

case 2:

deq();

break;

case 3:

e = frontelement();

if (e != 0)

printf("Front element : %d", e);

else

printf("\n No front element in Queue as queue is empty");

break;

case 4:

empty();

break;

case 5:

exit(0);

case 6:

display();

break;

case 7:

queuesize();

break;

default:

printf("Wrong choice, Please enter correct choice ");

break;

}

}

}

void create()

{

front = rear = NULL;

}

void queuesize()

{

printf("\n Queue size : %d", count);

}

void enq(int data)

{

if (rear == NULL)

{

rear = (struct node \*)malloc(1\*sizeof(struct node));

rear->ptr = NULL;

rear->info = data;

front = rear;

}

else

{

temp=(struct node \*)malloc(1\*sizeof(struct node));

rear->ptr = temp;

temp->info = data;

temp->ptr = NULL;

rear = temp;

}

count++;

}

void display()

{

front1 = front;

if ((front1 == NULL) && (rear == NULL))

{

printf("Queue is empty");

return;

}

while (front1 != rear)

{

printf("%d ", front1->info);

front1 = front1->ptr;

}

if (front1 == rear)

printf("%d", front1->info);

}

void deq()

{

front1 = front;

if (front1 == NULL)

{

printf("\n Error: Trying to display elements from empty queue");

return;

}

else

if (front1->ptr != NULL)

{

front1 = front1->ptr;

printf("\n Dequed value : %d", front->info);

free(front);

front = front1;

}

else

{

printf("\n Dequed value : %d", front->info);

free(front);

front = NULL;

rear = NULL;

}

count--;

}

int frontelement()

{

if ((front != NULL) && (rear != NULL))

return(front->info);

else

return 0;

}

void empty()

{

if ((front == NULL) && (rear == NULL))

printf("\n Queue empty");

else

printf("Queue not empty");

}