/\*EXPT 1 Develop a simple calculator to accept two floating point numbers from the keyboard.

Then display a menu to the user and let him/her select a mathematical operation to be

performed on those two numbers\*/

#include <stdio.h>

int main() {

float num1, num2, result;

int choice;

// Input two floating-point numbers from the user

printf("Enter number 1: ");

scanf("%f", &num1);

printf("Enter number 2: ");

scanf("%f", &num2);

// Display the menu

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

printf("Calculator Operations\n");

printf("1 - Add\n");

printf("2 - Subtract\n");

printf("3 - Multiply\n");

printf("4 - Divide\n");

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

// Input the choice from the user

printf("Enter your choice (1/2/3/4): ");

scanf("%d", &choice);

// Perform the selected operation and display the result

switch (choice) {

case 1:

result = num1 + num2;

printf("Result: %.2f\n", result);

break;

case 2:

result = num1 - num2;

printf("Result: %.2f\n", result);

break;

case 3:

result = num1 \* num2;

printf("Result: %.2f\n", result);

break;

case 4:

if (num2 == 0) {

printf("Error: Cannot divide by zero.\n");

} else {

result = num1 / num2;

printf("Result: %.2f\n", result);

}

break;

default:

printf("Invalid choice.\n");

}

return 0;

}

/\*2. Consider a currency system in which there are notes of seven denominations, namely,

Re.1, Rs. 2, Rs. 5, Rs.10, Rs. 50 and Rs. 100. If a sum of Rs. N is entered through the

keyboard, write a program to compute the smallest number of notes that will combine to

give Rs. N.\*/

#include <stdio.h>

int main() {

int denom[] = {100, 50, 10, 5, 2, 1};

int no\_notes[sizeof(denom) / sizeof(denom[0])];

int total\_amt;

printf("Enter the total amount in Rs.: ");

scanf("%d", &total\_amt);

printf("\nSmallest number of notes:\n");

for (int i = 0; i < sizeof(denom) / sizeof(denom[0]); i++) {

no\_notes[i] = total\_amt / denom[i];

total\_amt %= denom[i];

if (no\_notes[i] > 0) {

printf("%d x Rs. %d\n", no\_notes[i], denom[i]);

}

}

return 0;

}

/\*3. Write a program to find the roots (real and imaginary) of the quadratic equation of the

form ax2 +bx+c = 0.\*/

#include<stdio.h>

#include<math.h>

int main()

{

float a,b,c;

float discriminant,real,img,root1,root2;

printf("Enter the coefficients of the quadratic equation (a, b, c): ");

scanf("%f %f %f", &a, &b, &c);

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

//when rootsa are real and distinct

if (discriminant>0){

root1=(-b+ sqrt(discriminant))/(2\*a);

root2=(-b- sqrt(discriminant))/(2\*a);

printf("Roots are real and distinct\n ");

printf("Root 1 = %.2lf\n", root1);

printf("Root 2 = %.2lf\n", root2);

}

//when roots are equal

else if(discriminant==0.0){

double root = -b / (2 \* a);

printf("Roots are real and equal:\n");

printf("Root = %.2lf\n", root);

}

else{

//imaginary roots

real=-b/(2\*a);

img=sqrt(-discriminant)/(2\*a);

printf("Roots are imaginary:\n");

printf("Root 1 = %.2lf + %.2lfi\n", real, img);

printf("Root 2 = %.2lf - %.2lfi\n", real, img);

}

return 0;

}

/\*4. Write a program to input a student’s score in a subject (out of hundred) and print a grade

output. Use ternary operator and no other control statement. Also for grading purpose

please use the VIT grading system.\*/

#include <stdio.h>

int main() {

float score;

// Input the student's score

printf("Enter the student's score (out of 100): ");

scanf("%f", &score);

// Determine the grade using the VIT grading system

char grade = (score >= 90) ? 'S' :

(score >= 80) ? 'A' :

(score >= 70) ? 'B' :

(score >= 60) ? 'C' :

(score >= 55) ? 'D' :

(score >= 50) ? 'E' :

'F';

// Print the grade

printf("Grade: %c\n", grade);

return 0;

}

/\*5. Write a program to calculate mean of N many numbers taken as input through keyboard.

NOTE: Please do not use array.\*/

#include <stdio.h>

int main() {

int N, num, sum = 0;

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

scanf("%d", &N);

printf("Enter %d numbers:\n", N);

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

scanf("%d", &num);

sum += num;

}

double mean = (double) sum / N;

printf("Mean of the %d numbers is: %.2lf\n", N, mean);

return 0;

}

/\*6. Write a menu-driven program to,

1. Calculate the factorial of a given number using recursion, 2. Check whether the number

is prime or not,

3. Check whether the number is a palindrome or not.

Use different functions for each of the above tasks and using switch statement.\*/

#include <stdio.h>

// Function to calculate factorial using recursion

int factorial(int n) {

if (n == 0 || n == 1) {

return 1;

} else {

return n \* factorial(n - 1);

}

}

// Function to check if a number is prime

int isPrime(int n) {

if (n <= 1) {

return 0; // Not prime

}

for (int i = 2; i \* i <= n; i++) {

if (n % i == 0) {

return 0; // Not prime

}

}

return 1; // Prime

}

// Function to check if a number is palindrome

int isPalindrome(int n) {

int original = n;

int reversed = 0;

while (n > 0) {

reversed = reversed \* 10 + n % 10;

n /= 10;

}

return original == reversed;

}

int main() {

int choice, num;

while (1) {

printf("\nMenu:\n");

printf("1. Calculate factorial\n");

printf("2. Check prime number\n");

printf("3. Check palindrome\n");

printf("4. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

printf("Enter a number: ");

scanf("%d", &num);

printf("Factorial of %d is: %d\n", num, factorial(num));

break;

case 2:

printf("Enter a number: ");

scanf("%d", &num);

if (isPrime(num)) {

printf("%d is a prime number.\n", num);

} else {

printf("%d is not a prime number.\n", num);

}

break;

case 3:

printf("Enter a number: ");

scanf("%d", &num);

if (isPalindrome(num)) {

printf("%d is a palindrome.\n", num);

} else {

printf("%d is not a palindrome.\n", num);

}

break;

case 4:

printf("Exiting program.\n");

return 0;

default:

printf("Invalid choice. Please select a valid option.\n");

}

}

return 0;

}

/\*7. Write a recursive function to calculate the GCD (greatest common divisor) / HCF (highest

common factor) of two numbers.\*/

#include <stdio.h>

// Recursive function to calculate GCD using Euclidean algorithm

int gcd(int a, int b) {

if (b == 0) {

return a;

}

return gcd(b, a % b);

}

int main() {

int num1, num2;

printf("Enter two numbers: ");

scanf("%d %d", &num1, &num2);

int result = gcd(num1, num2);

printf("GCD of %d and %d is: %d\n", num1, num2, result);

return 0;

}

/\*8. Write a recursive function to print the first n Fibonacci numbers.\*/

#include <stdio.h>

// Recursive function to calculate and print Fibonacci numbers

void printFibonacci(int n, int a, int b) {

if (n <= 0) {

return;

}

int next = a + b;

printf("%d ", next);

printFibonacci(n - 1, b, next);

}

int main() {

int n;

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

scanf("%d", &n);

if (n <= 0) {

printf("Invalid input. Please enter a positive integer.\n");

return 1;

}

if (n >= 1) {

printf("First %d Fibonacci numbers:\n", n);

printf("0 ");

}

if (n >= 2) {

printf("1 ");

}

printFibonacci(n - 2, 0, 1);

printf("\n");

return 0;

}

/\*Given 2 nos. N and R, find the value of NCR using recursion. C(N, R) = C(N-1, R-1) + C(N-1, R)\*/

#include <stdio.h>

int ncr\_rec(int N, int r) {

    if (r == 0 || r == N) {

        return 1;

    }

    return ncr\_rec(N - 1, r - 1) + ncr\_rec(N - 1, r);

}

int main() {

    int N = 9;

    int r = 3;

    int result = ncr\_rec(N, r);

    printf("C(%d, %d) = %d\n", N, r, result);

    return 0;

}

/\*Use recursion to solve: 1+1/2+1/4+...+1/(2^N)\*/

#include <stdio.h>

double sum\_series(int N) {

    if (N == 0) {

        return 1.0;  // Base case: 1/(2^0) = 1

    }

    return 1.0 / (1 << N) + sum\_series(N - 1);

}

int main() {

    int N = 7;

    double result = sum\_series(N);

    printf("Sum of the series: %lf\n", result);

    return 0;

}

/\*WAP to swap contents of 2 variables using pointers without using the 3rd variable\*/

#include <stdio.h>

void swap(int \*a,int \*b){

    \*a=\*a+\*b;

    \*b=\*a-\*b;

    \*a=\*a-\*b;

}

int main(){

    int num1,num2,num3,num4;

    printf("enter number1 val:");

    scanf("%d",&num1);

    printf("enter number2 val:");

    scanf("%d",&num2);

    printf("Before the swap :-\n num1= %d\n num2= %d\n",num1,num2);

    swap(&num1,&num2);

    printf("After swapping them :-\n num1= %d\n num2= %d\n",num1,num2);

    return 0;

}

/\*Find the GCD of 2 numbers using Tail Recursion\*/

#include<stdio.h>

int gcd\_tailrec(int a ,int b){

    if(b==0){

        return a;

    }

    return gcd\_tailrec(b,a%b);

}

int main()

{

    int num1,num2;

    printf("Enter 2 numbers :");

    scanf("%d%d",&num1,&num2);

    if(num1<0||num2<0){

        printf("enter non-negative numbers \n");}

    else{

        int gcd = gcd\_tailrec(num1,num2);

        printf("GCD of %d and %d is %d\n",num1,num2,gcd);

    }

    return 0;

}

/\*WAP to create two 2-D arrays dynamically and then based on user choice do the following:

1. Find the sum of 2 matrices

2. Find the difference of the 2 matrices

3. Find the product of 2 matrices

4. Find the transpose of 2 matrices

Note:1. Create a header file matrices. h to declare the functions and use separate \*.c files to implement each functionality.

2. Each function should take pointers and other requisite data as parameters and should return an appropriate pointer to the main function.

\*/

MATRICES.H

#ifndef MATRICES\_H

#define MATRICES\_H

int\*\* createMatrix(int rows, int cols);

void freeMatrix(int\*\* matrix, int rows);

int\*\* matrixSum(int\*\* matrix1, int\*\* matrix2, int rows, int cols);

int\*\* matrixDifference(int\*\* matrix1, int\*\* matrix2, int rows, int cols);

int\*\* matrixProduct(int\*\* matrix1, int\*\* matrix2, int rows1, int cols1, int cols2);

int\*\* matrixTranspose(int\*\* matrix, int rows, int cols);

#endif

MAIN.C

#include <stdio.h>

#include "matrices.h"

// Function to input matrix elements

void inputMatrixElements(int\*\* matrix, int rows, int cols) {

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

for (int j = 0; j < cols; j++) {

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

}

}

}

void outputMatrixElements(int\*\* result, int rows, int cols) {

// Display the result matrix

printf("Result matrix:\n");

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

for (int j = 0; j < cols; j++) {

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

}

printf("\n");

}

}

void outputMatrixElements2(int\*\* transposedMatrix1,int\*\* transposedMatrix2, int rows, int cols) {

printf("Transposed Matrix 1:\n");

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

for (int j = 0; j < rows; j++) {

printf("%d ", transposedMatrix1[i][j]);

}

printf("\n");

}

printf("Transposed Matrix 2:\n");

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

for (int j = 0; j < rows; j++) {

printf("%d ", transposedMatrix2[i][j]);

}

printf("\n");

}

}

int main() {

int rows, cols;

int flag=0;

int choice;

printf("Enter the number of rows and columns for the matrices: ");

scanf("%d %d", &rows, &cols);

int\*\* matrix1 = createMatrix(rows, cols);

int\*\* matrix2 = createMatrix(rows, cols);

// input matrix elements

printf("Enter elements for matrix1:\n");

inputMatrixElements(matrix1, rows, cols);

printf("Enter elements for matrix2:\n");

inputMatrixElements(matrix2, rows, cols);

// Perform operations based on user choice

while(flag==0)

{

printf("Choose operation:\n");

printf("1. Sum\n2. Difference\n3. Product\n4. Transpose\n");

scanf("%d", &choice);

int\*\* result = NULL;

switch (choice) {

case 1:

result = matrixSum(matrix1, matrix2, rows, cols);

outputMatrixElements(result,rows,cols);

break;

case 2:

result = matrixDifference(matrix1, matrix2, rows, cols);

outputMatrixElements(result,rows,cols);

break;

case 3:

// Check if cols of matrix1 match rows of matrix2

if (cols != rows) {

printf("Matrix multiplication is not possible.\n");

break;

}

result = matrixProduct(matrix1, matrix2, rows, cols, cols);

outputMatrixElements(result,rows,cols);

break;

case 4:

int\*\* transposedMatrix1 = matrixTranspose(matrix1, rows, cols);

int\*\* transposedMatrix2 = matrixTranspose(matrix2, rows, cols);

outputMatrixElements2(transposedMatrix1,transposedMatrix2,rows,cols);

// Free memory for the transposed matrices

freeMatrix(transposedMatrix1, cols);

freeMatrix(transposedMatrix2, cols);

break;

default:

printf("Invalid choice.\n");

}

printf("do you want to continue?(0=yes||1==no)");

scanf("%d",flag);

}

// Free allocated memory

freeMatrix(matrix1, rows);

freeMatrix(matrix2, rows);

return 0;

}

CREATE\_MATRIX.C

#include <stdio.h>

#include <stdlib.h>

#include "matrices.h"

int\*\* createMatrix(int rows, int cols) {

int\*\* matrix = (int\*\*)malloc(rows \* sizeof(int\*));

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

matrix[i] = (int\*)malloc(cols \* sizeof(int));

}

return matrix;

}

void freeMatrix(int\*\* matrix, int rows) {

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

free(matrix[i]);

}

free(matrix);

}

SUM.C

#include "matrices.h"

int\*\* matrixSum(int\*\* matrix1, int\*\* matrix2, int rows, int cols) {

    int\*\* result = createMatrix(rows, cols);

    for (int i = 0; i < rows; i++) {

        for (int j = 0; j < cols; j++) {

            result[i][j] = matrix1[i][j] + matrix2[i][j];

        }

    }

    return result;

}

DIFF.C

#include "matrices.h"

int\*\* matrixDifference(int\*\* matrix1, int\*\* matrix2, int rows, int cols) {

    int\*\* result = createMatrix(rows, cols);

    for (int i = 0; i < rows; i++) {

        for (int j = 0; j < cols; j++) {

            result[i][j] = matrix1[i][j] - matrix2[i][j];

        }

    }

    return result;

}

MUL.C

#include "matrices.h"

int\*\* matrixProduct(int\*\* matrix1, int\*\* matrix2, int rows1, int cols1, int cols2) {

    int\*\* result = createMatrix(rows1, cols2);

    for (int i = 0; i < rows1; i++) {

        for (int j = 0; j < cols2; j++) {

            result[i][j] = 0;

            for (int k = 0; k < cols1; k++) {

                result[i][j] += matrix1[i][k] \* matrix2[k][j];

            }

        }

    }

    return result;

}

TRANSPOSE.C

#include "matrices.h"

int\*\* matrixTranspose(int\*\* matrix, int rows, int cols) {

    int\*\* result = createMatrix(cols, rows);

    for (int i = 0; i < rows; i++) {

        for (int j = 0; j < cols; j++) {

            result[j][i] = matrix[i][j];

        }

    }

    return result;

}

void transposeBothMatrices(int\*\* matrix1, int\*\* matrix2, int rows, int cols) {

    int\*\* transposedMatrix1 = matrixTranspose(matrix1, rows, cols);

    int\*\* transposedMatrix2 = matrixTranspose(matrix2, rows, cols);

    // Free memory for the transposed matrices

    freeMatrix(transposedMatrix1, cols);

    freeMatrix(transposedMatrix2, cols);

}

/\*expt 2-Create an employee structure which is capable of holding the information of the employees of an organization like name, id, designation, salary, etc. Consider that the organization has n such employees where n can be significantly large. Implement a program in C which uses functions for the following purpose:

1. To find the details of the highest paid employee

2. To find the details of the lowest paid employee

3. To return the mean salary offered by the organization.

\*/

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

struct Emp {

char name[50];

int id;

char desig[50];

float salary;

};

void highestPaidEmp(struct Emp \*employees, int n);

void lowestPaidEmp(struct Emp \*employees, int n);

float meanSalary(struct Emp \*employees, int n);

int main() {

int n;

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

scanf("%d", &n);

struct Emp \*employees = (struct Emp \*)malloc(n \* sizeof(struct Emp));

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

printf("Enter details for employee %d:\n", i + 1);

printf("Name: ");

scanf("%s", employees[i].name);

printf("ID: ");

scanf("%d", &employees[i].id);

printf("Designation: ");

scanf("%s", employees[i].desig);

printf("Salary: ");

scanf("%f", &employees[i].salary);

}

highestPaidEmp(employees, n);

lowestPaidEmp(employees, n);

float mean = meanSalary(employees, n);

printf("Mean salary offered by the organization: %.2f\n", mean);

free(employees);

return 0;

}

void highestPaidEmp(struct Emp \*employees, int n) {

int highestIndex = 0;

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

if (employees[i].salary > employees[highestIndex].salary) {

highestIndex = i;

}

}

printf("Details of the highest paid employee:\n");

printf("Name: %s\n", employees[highestIndex].name);

printf("ID: %d\n", employees[highestIndex].id);

printf("Designation: %s\n", employees[highestIndex].desig);

printf("Salary: %.2f\n", employees[highestIndex].salary);

}

void lowestPaidEmp(struct Emp \*employees, int n) {

int lowestIndex = 0;

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

if (employees[i].salary < employees[lowestIndex].salary) {

lowestIndex = i;

}

}

printf("Details of the lowest paid employee:\n");

printf("Name: %s\n", employees[lowestIndex].name);

printf("ID: %d\n", employees[lowestIndex].id);

printf("Designation: %s\n", employees[lowestIndex].desig);

printf("Salary: %.2f\n", employees[lowestIndex].salary);

}

float meanSalary(struct Emp \*employees, int n) {

float totalSalary = 0;

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

totalSalary += employees[i].salary;

}

return totalSalary / n;

/\*POINT RECTANGLE  PROBLEM\*/

#include <stdio.h>

#include <math.h>

// Structure for a Point

struct Point {

int x;

int y;

};

// Structure for a Line

struct Line {

struct Point start;

struct Point end;

float len;

};

// Function to calculate the length of a Line

float calculateLength(struct Line \*line) {

int dx = line->end.x - line->start.x;

int dy = line->end.y - line->start.y;

return sqrt(dx \* dx + dy \* dy);

}

int main() {

struct Line lines[4];

// Input for endpoints of four lines

for (int i = 0; i < 4; i++) {

printf("Enter the coordinates of the start point for Line %d (x y): ",

i + 1);

scanf("%d %d", &lines[i].start.x, &lines[i].start.y);

printf("Enter the coordinates of the end point for Line %d (x y): ", i

+ 1);

scanf("%d %d", &lines[i].end.x, &lines[i].end.y);

// Calculate the length of the line

lines[i].len = calculateLength(&lines[i]);

}

// Check if they can form a rectangle

int equalLengthCount = 0;

int hasEqualEndpoints = 0;

for (int i = 0; i < 4; i++) {

for (int j = i + 1; j < 4; j++) {

if (lines[i].len == lines[j].len) {

equalLengthCount++;

}

if ((lines[i].start.x == lines[j].start.x && lines[i].start.y ==

lines[j].start.y) &&

(lines[i].end.x == lines[j].end.x && lines[i].end.y ==

lines[j].end.y)) {

hasEqualEndpoints = 1;

}

}

}

// Determine the result

if (hasEqualEndpoints) {

printf("The lines can form a rectangle right away.\n");

} else if (equalLengthCount == 2) {

printf("The lines can be reoriented to form a rectangle.\n");

} else {

printf("The lines cannot form a rectangle.\n");

}

return 0;

/\*QUESTION: write a c program using linked lists to insert : 1)at the beginning 2)insert at the end

3)insert intermediate (a. nth position b. after node having value n c. before the node having value n)\*/

CODE:

#include <stdio.h>

#include <stdlib.h>

// Define a node structure

struct Node {

int data;

struct Node\* next;

};

// Function declarations

struct Node\* createNode(int data);

struct Node\* insertAtBeginning(struct Node\* head, int data);

struct Node\* insertAtEnd(struct Node\* head, int data);

struct Node\* insertAtNthPosition(struct Node\* head, int data, int position);

struct Node\* insertAfterValue(struct Node\* head, int data, int value);

struct Node\* insertBeforeValue(struct Node\* head, int data, int value);

void display(struct Node\* head);

int main() {

struct Node\* head = NULL;

char choice;

do {

int operation;

int data, position, value;

printf("\nChoose an operation:\n");

printf("1. Insert at the beginning\n");

printf("2. Insert at the end\n");

printf("3. Insert at nth position\n");

printf("4. Insert after a specific value\n");

printf("5. Insert before a specific value\n");

printf("6. Display the list\n");

printf("7. Exit\n");

printf("Enter your choice: ");

scanf("%d", &operation);

switch (operation) {

case 1:

printf("Enter data to insert at the beginning: ");

scanf("%d", &data);

head = insertAtBeginning(head, data);

break;

case 2:

printf("Enter data to insert at the end: ");

scanf("%d", &data);

head = insertAtEnd(head, data);

break;

case 3:

printf("Enter data to insert: ");

scanf("%d", &data);

printf("Enter position to insert at: ");

scanf("%d", &position);

head = insertAtNthPosition(head, data, position);

break;

case 4:

printf("Enter data to insert: ");

scanf("%d", &data);

printf("Enter the value after which to insert: ");

scanf("%d", &value);

head = insertAfterValue(head, data, value);

break;

case 5:

printf("Enter data to insert: ");

scanf("%d", &data);

printf("Enter the value before which to insert: ");

scanf("%d", &value);

head = insertBeforeValue(head, data, value);

break;

case 6:

printf("Linked List: ");

display(head);

break;

case 7:

printf("Exiting program.\n");

exit(0);

default:

printf("Invalid choice. Please try again.\n");

}

printf("Do you want to perform another operation? (y/n): ");

scanf(" %c", &choice);

} while (choice == 'y' || choice == 'Y');

return 0;

}

// Function to create a new node

struct Node\* createNode(int data) {

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

if (newNode == NULL) {

printf("Memory allocation failed.\n");

exit(1);

}

newNode->data = data;

newNode->next = NULL;

return newNode;

}

// Function to insert a node at the beginning of the list

struct Node\* insertAtBeginning(struct Node\* head, int data) {

struct Node\* newNode = createNode(data);

newNode->next = head;

return newNode;

}

// Function to insert a node at the end of the list

struct Node\* insertAtEnd(struct Node\* head, int data) {

struct Node\* newNode = createNode(data);

if (head == NULL) {

return newNode;

}

struct Node\* current = head;

while (current->next != NULL) {

current = current->next;

}

current->next = newNode;

return head;

}

// Function to insert a node at the nth position in the list

struct Node\* insertAtNthPosition(struct Node\* head, int data, int position) {

if (position < 1) {

printf("Invalid position.\n");

return head;

}

if (position == 1) {

return insertAtBeginning(head, data);

}

struct Node\* newNode = createNode(data);

struct Node\* current = head;

int currentPosition = 1;

while (current != NULL && currentPosition < position - 1) {

current = current->next;

currentPosition++;

}

if (current == NULL) {

printf("Invalid position.\n");

return head;

}

newNode->next = current->next;

current->next = newNode;

return head;

}

// Function to insert a node after a node with a specific value

struct Node\* insertAfterValue(struct Node\* head, int data, int value) {

struct Node\* newNode = createNode(data);

struct Node\* current = head;

while (current != NULL) {

if (current->data == value) {

newNode->next = current->next;

current->next = newNode;

return head;

}

current = current->next;

}

printf("Value not found in the list.\n");

return head;

}

// Function to insert a node before a node with a specific value

struct Node\* insertBeforeValue(struct Node\* head, int data, int value) {

struct Node\* newNode = createNode(data);

if (head == NULL) {

printf("List is empty.\n");

return head;

}

if (head->data == value) {

newNode->next = head;

return newNode;

}

struct Node\* current = head;

while (current->next != NULL) {

if (current->next->data == value) {

newNode->next = current->next;

current->next = newNode;

return head;

}

current = current->next;

}

printf("Value not found in the list.\n");

return head;

}

// Function to display the linked list

void display(struct Node\* head) {

struct Node\* current = head;

while (current != NULL) {

printf("%d -> ", current->data);

current = current->next;

}

printf("NULL\n");

}

/\*Q1. Code own printf function without using printf function\*/

#include <stdio.h>

#include <stdarg.h>

// Function declaration

void custom\_printf(const char \*format, ...);

int main() {

int age;

char name[100];

printf("Enter name: ");

scanf("%s", name);

printf("Enter age: ");

scanf("%d", &age);

// Call custom\_printf to display the user input

custom\_printf("Name: %s, Age: %d\n", name, age);

return 0;

}

// Function definition

void custom\_printf(const char \*format, ...) {

va\_list args;

va\_start(args, format);

while (\*format) {

if (\*format == '%') {

format++;

if (\*format == 'd') {

int num = va\_arg(args, int);

// Convert the integer to a string for output

char num\_str[12]; // Assuming a maximum of 11 digits

snprintf(num\_str, sizeof(num\_str), "%d", num);

for (int i = 0; num\_str[i]; i++) {

putchar(num\_str[i]);

}

} else if (\*format == 's') {

char \*str = va\_arg(args, char\*);

// Output each character of the string

while (\*str) {

putchar(\*str);

str++;

}

} else {

putchar('%');

putchar(\*format);

}

} else {

putchar(\*format);

}

format++;

}

va\_end(args);}

/\*Q2. Consider a file a1. C which has the names of employees of an organization. Write a program in C

to fetch the names of each employee from a1. C and take their age and salary as input from the end

user. Consider a structure to hold the information and then create a file b1. C to write the name age

and salary of each employee into it.\*/

#include <stdio.h>

#include <stdlib.h>

struct Employee {

char name[150];

int age;

float salary;

};

int main() {

FILE \*inputfile = fopen("a1.c", "r");

FILE \*outputfile = fopen("b1.c", "w");

if (inputfile == NULL || outputfile == NULL) {

printf("File opening failed.\n");

exit(1);

}

struct Employee employee;

while (fscanf(inputfile, "%s", employee.name) == 1) {

printf("Enter age of %s: ", employee.name);

scanf("%d", &employee.age);

printf("Enter salary of %s: ", employee.name);

scanf("%f", &employee.salary);

fprintf(outputfile, "Name: %s\nAge: %d\nSalary: %.2f\n\n",

employee.name, employee.age, employee.salary);

}

fclose(inputfile);

fclose(outputfile);

printf("Data is successfully written into b1.c.\n");

return 0;

}

/\*write a menu driven program for the following

a)find the absolute value of a number

b)find the sin cos and tan of 0,45 90 135 and 180 degrees

then use the sin^-1,cos^-1and tan^-1 to get the old standard values back

c)find the ceiling ,floor and round of a floating point no taken as input

d)find the time taken to execute a code snippet

eg: for(i=0;i<0xffer;i++)

sum+=i;

code:\*/

#include <stdio.h>

#include <math.h>

#include <time.h>

#define PI 3.14159265358979323846

void angleoperation(double angle);

void timetaken();

int main() {

int choice;

double num;

while (1) {

printf("1. Find the absolute value of a number\n");

printf("2. Find sin, cos, and tan of 0, 45, 90, 135, and 180

degrees\n");

printf("3. Find ceiling, floor, and round of a float number\n");

printf("4. Measure execution time\n");

printf("5. Exit\n");

printf("Enter yor choice please: ");

scanf("%d", &choice);

switch (choice) {

case 1:

printf("Enter number: ");

scanf("%lf", &num);

printf("Absolute value: %.2lf\n", fabs(num));

break;

case 2:

printf("Angle (0, 45, 90, 135, 180 degrees): ");

scanf("%lf", &num);

angleoperation(num);

break;

case 3:

printf("Enter a floating-point number: ");

scanf("%lf", &num);

printf("Ceiling: %.2lf\n", ceil(num));

printf("Floor: %.2lf\n", floor(num));

printf("Rounded value: %.2lf\n", round(num));

break;

case 4:

timetaken();

break;

case 5:

return 0;

default:

printf("Invalid choice. Please try again.\n");

}

}

return 0;

}

void angleoperation(double angle) {

double radians = angle \* PI / 180; // Convert to radians using the custom

PI constant

double a=sin(radians);

double b=cos(radians);

double c=tan(radians);

double ia=asin(a) \* 180 / PI;

double ib=acos(b) \* 180 / PI;

double ic=atan(c) \* 180 / PI;

printf("Sine: %.2lf\n", a);

printf("Cosine: %.2lf\n", b);

printf("Tangent: %.2lf\n",c);

// Adjust results if they fall outside of expected range

if (angle == 180 ) {

ia +=180;

ic +=180;

}

if (angle == 135) {

ia =180-ia;

ic =180+ic;

}

// Calculate inverse trigonometric values

printf("Recovering original value in degrees using inverse functions

:\n");

printf("via Inverse Sine: %.2lf degrees\n", ia);

printf("via Inverse Cosine: %.2lf degrees\n", ib);

printf("via Inverse Tangent: %.2lf degrees\n",ic);

}

void timetaken(){

clock\_t start, end;

double cpu\_time\_used;

int sum=0;

start = clock();

for (int i = 0; i < 55555555; i++) {

sum+=i;

}

end = clock();

cpu\_time\_used = ((double)(end - start)) / CLOCKS\_PER\_SEC;

printf("Time taken to execute code: %lf seconds\n",

cpu\_time\_used);

}

/\*1. Write a program in C to add 2 polynomials using arrays.

Case 1. Use 3 arrays\*/

Code :

#include<stdio.h>

#include<stdlib.h>

//Structure to represent a polynomial

struct Polynomial {

int degree;

int coefficients[100]; // Assuming a maximum degree of 99

};

// Function declarations

void inputPolynomial(struct Polynomial\* poly);

void addPolynomials(const struct Polynomial\* poly1, const struct Polynomial\* poly2, struct Polynomial\* result);

void displayPolynomial(const struct Polynomial\* poly);

int main() {

int numPolynomials;

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

scanf("%d", &numPolynomials);

struct Polynomial polynomials[numPolynomials];

struct Polynomial sum;

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

printf("Enter the details for polynomial %d:\n", i + 1);

inputPolynomial(&polynomials[i]);

}

// Initialize the sum with the first polynomial

sum = polynomials[0];

for (int i = 1; i < numPolynomials; i++) {

struct Polynomial temp = sum;

addPolynomials(&temp, &polynomials[i], &sum);

}

printf("Sum of the polynomials is: ");

displayPolynomial(&sum);

return 0;

}

// Function definitions

void inputPolynomial(struct Polynomial\* poly) {

printf("Enter the degree of the polynomial: ");

scanf("%d", &poly->degree);

for (int i = poly->degree; i >= 0; i--) {

printf("Enter the coefficient for x^%d: ", i);

scanf("%d", &poly->coefficients[i]);

}

}

void addPolynomials(const struct Polynomial\* poly1, const struct Polynomial\* poly2, struct Polynomial\* result) {

int maxDegree = poly1->degree > poly2->degree ? poly1->degree : poly2->degree;

result->degree = maxDegree;

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

result->coefficients[i] = poly1->coefficients[i] + poly2->coefficients[i];

}

}

void displayPolynomial(const struct Polynomial\* poly) {

for (int i = poly->degree; i > 0; i--) {

if (poly->coefficients[i] != 0) {

printf("%dx^%d + ", poly->coefficients[i], i);

}

}

printf("%d\n", poly->coefficients[0]);

}

Output

/\*Case 2. Use only 1 array (efficient memory usage)

Code 2:\*/

#include <stdio.h>

// Structure to represent a polynomial

struct Polynomial {

int degree;

int coefficients[100]; // Assuming a maximum degree of 99

};

// Function declarations

void inputPolynomial(struct Polynomial\* poly);

void addPolynomials(const struct Polynomial\* poly1, const struct Polynomial\* poly2, struct Polynomial\* result);

void displayPolynomial(const struct Polynomial\* poly);

int main() {

int numPolynomials;

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

scanf("%d", &numPolynomials);

struct Polynomial polynomials[numPolynomials];

struct Polynomial sum;

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

printf("Enter the details for polynomial %d:\n", i + 1);

inputPolynomial(&polynomials[i]);

}

// Initialize the sum with the first polynomial

sum = polynomials[0];

for (int i = 1; i < numPolynomials; i++) {

struct Polynomial temp = sum;

addPolynomials(&temp, &polynomials[i], &sum);

}

printf("Sum of the polynomials is: ");

displayPolynomial(&sum);

return 0;

}

// Function definitions

void inputPolynomial(struct Polynomial\* poly) {

printf("Enter the degree of the polynomial: ");

scanf("%d", &poly->degree);

for (int i = poly->degree; i >= 0; i--) {

printf("Enter the coefficient for x^%d: ", i);

scanf("%d", &poly->coefficients[i]);

}

}

void addPolynomials(const struct Polynomial\* poly1, const struct Polynomial\* poly2, struct Polynomial\* result) {

int maxDegree = poly1->degree > poly2->degree ? poly1->degree : poly2->degree;

result->degree = maxDegree;

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

result->coefficients[i] = poly1->coefficients[i] + poly2->coefficients[i];

}

}

void displayPolynomial(const struct Polynomial\* poly) {

for (int i = poly->degree; i > 0; i--) {

if (poly->coefficients[i] != 0) {

printf("%dx^%d + ", poly->coefficients[i], i);

}

}

printf("%d\n", poly->coefficients[0]);

}

Output :

/\*2. Write a program in C to multiply 2 polynomials using arrays.\*/

Code:

#include <stdio.h>

// Structure to represent a polynomial

struct Polynomial {

int degree;

int coefficients[100]; // Assuming a maximum of 100 coefficients

};

// Function declarations

void inputPolynomial(struct Polynomial \*poly);

void multiplyPolynomials(const struct Polynomial \*poly1, const struct Polynomial \*poly2, struct Polynomial

\*result);

void displayPolynomial(const struct Polynomial \*poly);

int main() {

struct Polynomial poly1, poly2, result;

printf("Enter the first polynomial:\n");

inputPolynomial(&poly1);

printf("Enter the second polynomial:\n");

inputPolynomial(&poly2);

multiplyPolynomials(&poly1, &poly2, &result);

printf("Product of the two polynomials is: ");

displayPolynomial(&result);

return 0;

}

// Function definitions

void inputPolynomial(struct Polynomial \*poly) {

printf("Enter the degree of the polynomial: ");

scanf("%d", &poly->degree);

for (int i = poly->degree; i >= 0; i--) {

printf("Enter the coefficient for x^%d: ", i);

scanf("%d", &poly->coefficients[i]);

}

}

void multiplyPolynomials(const struct Polynomial \*poly1, const struct Polynomial \*poly2, struct Polynomial

\*result) {

result->degree = poly1->degree + poly2->degree;

for (int i = 0; i <= result->degree; i++) {

result->coefficients[i] = 0; // Initialize the result polynomial coefficients to 0

}

for (int i = 0; i <= poly1->degree; i++) {

for (int j = 0; j <= poly2->degree; j++) {

result->coefficients[i + j] += poly1->coefficients[i] \* poly2->coefficients[j];

}

}

}

void displayPolynomial(const struct Polynomial \*poly) {

for (int i = poly->degree; i >= 0; i--) {

if (poly->coefficients[i] != 0) {

if (i == 0) {

printf("%d\n", poly->coefficients[i]);

} else {

printf("%dx^%d + ", poly->coefficients[i], i);

}

}

}

}

/\*Q1) Demonstrate using makefile to build multiple .c files into a final application.

Defining a simple calculator application for the demo.

\*/

calculator.h

void add();

void subtract();

void multiply();

void divide();

/\*add.c\*/

#include "calculator.h"

#include <stdio.h> void add() {

double a, b;

printf("Enter 2 numbers: ");

scanf("%lf %lf", &a, &b);

printf("The sum is %f\n", a + b);

}

/\*sub.c\*/

#include "calculator.h"

#include <stdio.h>

void subtract()

{     double a, b;

printf("Enter 2 numbers: ");

scanf("%lf %lf", &a, &b);

printf("The difference is %f\n", a - b);

}

/\*mul.c \*/

#include "calculator.h"

 int multiply(int a, int b) {

    return a \* b; }

/\*div.c\*/

#include "calculator.h"

#include <stdio.h>

void multiply()

{

    double a, b;

    printf("Enter 2 numbers: ");

    scanf("%lf %lf", &a, &b);

    printf("The product is %f\n", a \* b);

}

/\*main.c\*/

#include <stdio.h>

#include "calculator.h"

int main()

{     add();

    subtract();

    multiply();

    divide();

}

makefile with macros

CC = gcc #compilation command

CFLAGS = -c

output: main.o add.o sub.o mul.o div.o

    $(CC) main.o add.o sub.o mul.o div.o -o output

add.o:

    $(CC) $(CFLAGS) add.c

mul.o:

    $(CC) $(CFLAGS) mul.c

div.o:

    $(CC) $(CFLAGS) div.c

sub.o:

    $(CC) $(CFLAGS) sub.c

main.o:

    $(CC) $(CFLAGS) main.c

clean:

    rm \*.o output building and running the program

If we try to remake, it shows output is up to date

If we remove the final output, and remake, only the final output is made again.

Using make clean

**Q1)** Demonstrate using makefile to build multiple .c files into a final application.

Defining a simple calculator application for the demo.

calculator.h void add(); void subtract(); void multiply(); void divide(); add.c

#include "calculator.h" #include <stdio.h> void add() { double a, b; printf("Enter 2 numbers: "); scanf("%lf %lf", &a, &b); printf("The sum is %f\n", a + b);

}

## sub.c

#include "calculator.h" #include <stdio.h> void subtract()

{ double a, b; printf("Enter 2 numbers: "); scanf("%lf %lf", &a, &b); printf("The difference is %f\n", a - b);

}

## mul.c

#include "calculator.h"

int multiply(int a, int b) { return a \* b; }

## div.c

#include "calculator.h" #include <stdio.h> void multiply()

{

double a, b; printf("Enter 2 numbers: "); scanf("%lf %lf", &a, &b); printf("The product is %f\n", a \* b);

}

## main.c

#include <stdio.h>

# #include "calculator.h"

int main()

{ add(); subtract(); multiply(); divide();

# }

## makefile with macros

CC = gcc #compilation command

CFLAGS = -c

output: main.o add.o sub.o mul.o div.o

$(CC) main.o add.o sub.o mul.o div.o -o output

add.o:

$(CC) $(CFLAGS) add.c

mul.o:

$(CC) $(CFLAGS) mul.c

div.o:

$(CC) $(CFLAGS) div.c

sub.o:

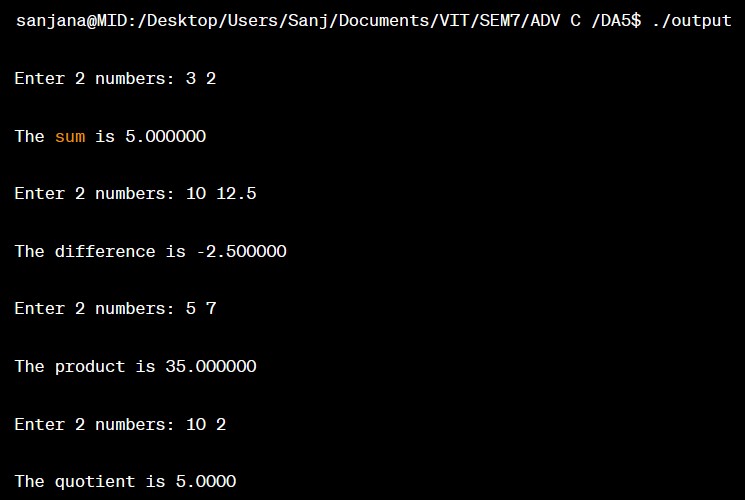
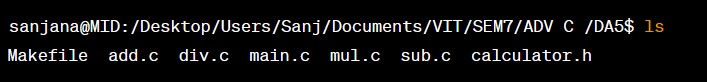
$(CC) $(CFLAGS) sub.c

# main.o:

$(CC) $(CFLAGS) main.c

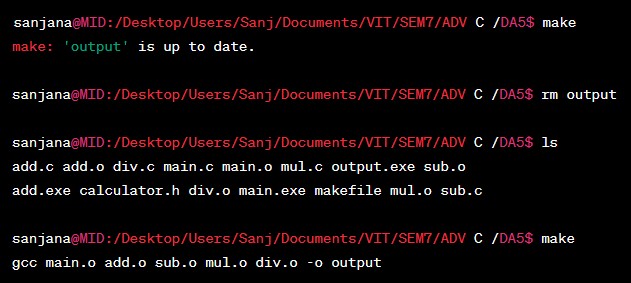
clean:

rm \*.o output building and running the program



If we try to remake, it shows output is up to date

If we remove the final output, and remake, only the final output is made again.



Using make clean

