FUNCTION

1. Write a C program to add, subtract, multiply and divide two integers using userdefined type function with return type.

#include <stdio.h>

// Function declarations

int add(int a, int b);

int subtract(int a, int b);

int multiply(int a, int b);

float divide(int a, int b);

int main() {

int num1, num2;

// Input two numbers from the user

printf("Enter first number: ");

scanf("%d", &num1);

printf("Enter second number: ");

scanf("%d", &num2);

// Perform operations and display results

printf("Sum: %d\n", add(num1, num2));

printf("Difference: %d\n", subtract(num1, num2));

printf("Product: %d\n", multiply(num1, num2));

if (num2 != 0) {

printf("Quotient: %.2f\n", divide(num1, num2));

} else {

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

}

return 0;

}

// Function definitions

int add(int a, int b) {

return a + b;

}

int subtract(int a, int b) {

return a - b;

}

int multiply(int a, int b) {

return a \* b;

}

float divide(int a, int b) {

return (float)a / b;

}

2. Write a C program to calculate sum of first 20 natural numbers using recursive function.

#include <stdio.h>

// Function declaration

int sumOfNaturals(int n);

int main() {

int n = 20;

// Calculate and print the sum of the first 20 natural numbers

printf("Sum of first %d natural numbers: %d\n", n, sumOfNaturals(n));

return 0;

}

// Function definition

int sumOfNaturals(int n) {

if (n == 1) {

return 1;

} else {

return n + sumOfNaturals(n - 1);

}

}

3. Write a C program to generate Fibonacci series using recursive function.

#include <stdio.h>

// Function declaration

int fibonacci(int n);

int main() {

int N;

// Input the number of terms in the Fibonacci series

printf("Enter the number of terms in the Fibonacci series: ");

scanf("%d", &N);

// Generate and print the Fibonacci series

printf("Fibonacci series up to %d terms: ", N);

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

printf("%d ", fibonacci(i));

}

return 0;

}

// Function definition

int fibonacci(int n) {

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

return n;

} else {

return fibonacci(n - 1) + fibonacci(n - 2);

}

}

4. Write a C program to swap two integers using call by value and call by reference methods of

passing arguments to a function.

#include <stdio.h>

// Function declarations

void swapByValue(int a, int b);

void swapByReference(int \*a, int \*b);

int main() {

int num1, num2;

// Input two numbers from the user

printf("Enter first number: ");

scanf("%d", &num1);

printf("Enter second number: ");

scanf("%d", &num2);

// Call function to swap by value

printf("Before swapping (by value): num1 = %d, num2 = %d\n", num1, num2);

swapByValue(num1, num2);

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

// Call function to swap by reference

printf("Before swapping (by reference): num1 = %d, num2 = %d\n", num1, num2);

swapByReference(&num1, &num2);

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

return 0;

}

// Function definition to swap by value

void swapByValue(int a, int b) {

int temp = a;

a = b;

b = temp;

}

// Function definition to swap by reference

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

int temp = \*a;

\*a = \*b;

\*b = temp;

}

5. Write a C program to find sum of digits of the number using Recursive Function.

#include <stdio.h>

// Function declaration

int sumOfDigits(int n);

int main() {

int num;

// Input number from the user

printf("Enter a number: ");

scanf("%d", &num);

// Calculate and print the sum of digits

printf("Sum of digits of %d is: %d\n", num, sumOfDigits(num));

return 0;

}

// Function definition

int sumOfDigits(int n) {

if (n == 0) {

return 0;

} else {

return (n % 10) + sumOfDigits(n / 10);

}

}

6. Write a C program to read an integer number and print the reverse of that number using recursion.

#include <stdio.h>

// Function declaration

int reverseNumber(int n);

int main() {

int num;

// Input number from the user

printf("Enter a number: ");

scanf("%d", &num);

// Calculate and print the reverse of the number

printf("Reverse of %d is: %d\n", num, reverseNumber(num));

return 0;

}

// Function definition

int reverseNumber(int n) {

static int reversedNumber = 0;

if (n == 0) {

return 0;

} else {

reversedNumber = reversedNumber \* 10 + n % 10;

reverseNumber(n / 10);

return reversedNumber;

}

}

7. Write a C program to find maximum and minimum between two numbers using functions.

#include <stdio.h>

// Function declarations

int findMaximum(int a, int b);

int findMinimum(int a, int b);

int main() {

int num1, num2;

// Input two numbers from the user

printf("Enter first number: ");

scanf("%d", &num1);

printf("Enter second number: ");

scanf("%d", &num2);

// Find and print the maximum and minimum

printf("Maximum between %d and %d is: %d\n", num1, num2, findMaximum(num1, num2));

printf("Minimum between %d and %d is: %d\n", num1, num2, findMinimum(num1, num2));

return 0;

}

// Function definition to find the maximum

int findMaximum(int a, int b) {

return (a > b) ? a : b;

}

// Function definition to find the minimum

int findMinimum(int a, int b) {

return (a < b) ? a : b;

}

8. Write a C program to check whether a number is even or odd using functions.

#include <stdio.h>

// Function declaration

int isEven(int n);

int main() {

int num;

// Input number from the user

printf("Enter a number: ");

scanf("%d", &num);

// Check and print whether the number is even or odd

if (isEven(num)) {

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

} else {

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

}

return 0;

}

// Function definition to check whether a number is even

int isEven(int n) {

return (n % 2 == 0);

}

9. Write a C program to check whether a number is prime, Armstrong or perfect number using functions.

#include <stdio.h>

#include <math.h>

// Function declarations

int isPrime(int n);

int isArmstrong(int n);

int isPerfect(int n);

int main() {

int num;

// Input number from the user

printf("Enter a number: ");

scanf("%d", &num);

// Check and print whether the number is prime, Armstrong, or perfect

if (isPrime(num)) {

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

} else {

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

}

if (isArmstrong(num)) {

printf("%d is an Armstrong number.\n", num);

} else {

printf("%d is not an Armstrong number.\n", num);

}

if (isPerfect(num)) {

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

} else {

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

}

return 0;

}

// Function definition to check whether 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 definition to check whether a number is Armstrong

int isArmstrong(int n) {

int originalNumber, remainder, digits = 0, result = 0;

originalNumber = n;

while (originalNumber != 0) {

originalNumber /= 10;

++digits;

}

originalNumber = n;

while (originalNumber != 0) {

remainder = originalNumber % 10;

result += pow(remainder, digits);

originalNumber /= 10;

}

return (result == n);

}

// Function definition to check whether a number is perfect

int isPerfect(int n) {

int sum = 1; // 1 is always a divisor

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

if (n % i == 0) {

sum += i;

if (i != n / i) {

sum += n / i;

}

}

}

return (sum == n);

}

10. Write a C program to find power of any number using recursion.

#include <stdio.h>

// Function declaration

double power(double base, int exponent);

int main() {

double base;

int exponent;

// Input base and exponent from the user

printf("Enter the base: ");

scanf("%lf", &base);

printf("Enter the exponent: ");

scanf("%d", &exponent);

// Calculate and print the power

printf("%.2lf^%d = %.2lf\n", base, exponent, power(base, exponent));

return 0;

}

// Function definition to calculate power using recursion

double power(double base, int exponent) {

if (exponent == 0) {

return 1;

} else if (exponent > 0) {

return base \* power(base, exponent - 1);

} else {

// Handling negative exponent

return 1 / (base \* power(base, -exponent - 1));

}

}