```
//WAP to implement a function which is going to add two number
#include <stdio.h>
void sum2Elements(int, int);
int main(){
     int a = 20, b = 30;
     sum2Elements(a, b);
    return 0;
}
/*
Name: fun()
Return Type: void
Parameter:(data type of each parameter): No parameters
Shord disciption: it is used to tract the number of times the
                function is getting called
*/

//function definition
void sum2Elements(int a, int b){
    int sum = 0;
    sum = a + b;
    printf("Sum = %d \n",sum);
}


//WAP to implement a function which is going to add two number
#include <stdio.h>
void sum2Elements(int, int);
int main(){
     int a = 20, b = 30;
     //call by value
     sum2Elements(a, b);
     printf("001a=%d and b=%d /n",a,b);
    return 0;
}
/*
Name: fun()
Return Type: void
Parameter:(data type of each parameter): No parameters
Shord disciption: it is used to tract the number of times the
                function is getting called
*/

//function definition
void sum2Elements(int e, int d){
    int a=30,b=40;
    printf("002e=%d and d=%d /n",a,b);
    int sum = 0;
    sum = e + d;

    printf("Sum = %d \n",sum);
}

1.//WAP to implement a function which is going to add two number
#include <stdio.h>
int sum2Elements(int, int);
int main(){
     int a = 20, b = 30;
     int sumMain=0;
     sumMain=sum2Elements(a, b);
     printf("sumMain = %d",sumMain);
     printf("001a=%d and b=%d \n",a,b);
    return 0;
}
/*
Name: fun()
Return Type: void
Parameter:(data type of each parameter): No parameters
Shord disciption: it is used to tract the number of times the
                function is getting called
*/

//function definition
int sum2Elements(int e, int d){
    int a=30,b=40;
    printf("002e=%d and d=%d \n",a,b);
    int sumMain = 0;
    sumMain = e + d;
    return sumMain;

}
```

2.Write a program to swap two numbers using a function. Observe and explain why the original numbers remain unchanged due to call by value.

```
#include <stdio.h>

// Function to swap two numbers
void swap_numbers(int a, int b) {
    int temp;
    temp = a;
    a = b;
    b = temp;
    printf("Inside function after swapping: a = %d, b = %d\n", a, b);
}

int main() {
    int num1 = 5, num2 = 10;

    // Print before calling the function
    printf("Before swapping: num1 = %d, num2 = %d\n", num1, num2);

    // Call the swap function
    swap_numbers(num1, num2);

    // Print after calling the function
    printf("After function call: num1 = %d, num2 = %d\n", num1, num2);

    return 0;
}
```

USING RETURN TYPE

```c
#include <stdio.h>

// Function to swap two numbers using pointers and return an integer value
int swap_numbers(int *a, int *b) {
    int temp;
    temp = *a;   // Dereference pointer a to get the value
    *a = *b;     // Dereference pointer b and assign to *a
    *b = temp;   // Assign temp (old value of *a) to *b
    return 1;    // Return 1 to indicate a successful swap
}

int main() {
    int num1 = 5, num2 = 10;

    // Print before calling the function
    printf("Before swapping: num1 = %d, num2 = %d\n", num1, num2);

    // Call the swap function and pass the addresses of num1 and num2
    swap_numbers(&num1, &num2);

    // Print after calling the function
    printf("After function call: num1 = %d, num2 = %d\n", num1, num2);

    return 0;
}
```

3.Implement a function that takes two integers as arguments and returns the larger of the two. Demonstrate how the original values are not altered

```c
#include <stdio.h>

// Function that returns the larger of two integers
int get_larger(int a, int b) {
    if (a > b) {
        return a;
    } else {
        return b;
    }
}

int main() {
    int num1 = 10;
    int num2 = 20;

    // Printing the original values before calling the function
    printf("Original values: num1 = %d, num2 = %d\n", num1, num2);

    // Get the larger number
    int larger = get_larger(num1, num2);

    // Print the result
    printf("Larger value: %d\n", larger);

    // Demonstrating that the original values are not altered
    printf("After function call: num1 = %d, num2 = %d\n", num1, num2);
    return 0;
}
```

WITHOUT RETURN TYPE

```
#include <stdio.h>

// Function that prints the larger of two integers
void print_larger(int a, int b) {
    if (a > b) {
        printf("Larger value: %d\n", a);
    } else {
        printf("Larger value: %d\n", b);
    }
}

int main() {
    int num1 = 10;
    int num2 = 20;

    // Printing the original values before calling the function
    printf("Original values: num1 = %d, num2 = %d\n", num1, num2);

    // Calling the function to print the larger number
    print_larger(num1, num2);

    // Demonstrating that the original values are not altered
    printf("After function call: num1 = %d, num2 = %d\n", num1, num2);

    return 0;
}
```

4.Create a function to compute the factorial of a given number passed to it. Ensure the original number remains unaltered.

```
#include <stdio.h>

// Function to compute the factorial of a given number
long long int factorial(int n) {
    long long int result = 1;
    for (int i = 1; i <= n; i++) {
        result *= i;
    }
    return result;
}

int main() {
    int num = 5;

    // The original number remains unaltered
    printf("Original number: %d\n", num);

    // Compute the factorial and print it
    printf("Factorial of %d is: %lld\n", num, factorial(num));

    return 0;
}
```

WITHOUT RETURN TYPE

```
#include <stdio.h>

// Function to compute the factorial of a given number and print it
void factorial(int n) {
    long long int result = 1;

    // Compute factorial using an iterative approach
    for (int i = 1; i <= n; i++) {
        result *= i;
    }

    // Print the result
    printf("Factorial of %d is: %lld\n", n, result);
}

int main() {
    int num = 5;

    // The original number remains unaltered
    printf("Original number: %d\n", num);

    // Call the factorial function to compute and print the factorial
    factorial(num);

    return 0;
}
```

5.Write a program where a function determines whether a given integer is even or odd. The function should use call by value

```
#include <stdio.h>

// Function to check if the number is even or odd
void checkEvenOdd(int num) {
    if (num % 2 == 0) {
        printf("%d is even.\n", num);
    } else {
        printf("%d is odd.\n", num);
    }
}

int main() {
    int number;

    // Ask user to input a number
    printf("Enter an integer: ");
    scanf("%d", &number);

    // Call the function to check if the number is even or odd
    checkEvenOdd(number);

    return 0;
}
```

WITH RETURN TYPE

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

// Function to check if the number is even or odd and return the result
// Returns 1 for even, 0 for odd
int checkEvenOdd(int num) {
    if (num % 2 == 0) {
        return 1; // Even
    } else {
        return 0; // Odd
    }
}

int main() {
    int number;
    int result;

    // Ask user to input a number
    printf("Enter an integer: ");
    scanf("%d", &number);

    // Call the function to check if the number is even or odd
    result = checkEvenOdd(number);

    // Output the result
    if (result == 1) {
        printf("%d is even.\n", number);
    } else {
        printf("%d is odd.\n", number);
    }

    return 0;
}
```

6.Write a program that calculates simple interest using a function. Pass principal, rate, and time as arguments and return the computed interest

```
#include <stdio.h>

// Function to calculate simple interest
float calculateSimpleInterest(float principal, float rate, float time) {
    return (principal * rate * time) / 100;
}

int main() {
    float principal, rate, time, interest;

    // Input values
    printf("Enter the principal amount: ");
    scanf("%f", &principal);

    printf("Enter the rate of interest: ");
    scanf("%f", &rate);
    printf("Enter the time period in years: ");
    scanf("%f", &time);

    // Calculate simple interest using the function
    interest = calculateSimpleInterest(principal, rate, time);

    // Output the result
    printf("The simple interest is: %.2f\n", interest);

    return 0;
}
```

WITHOUT RETURN TYPE

```
#include <stdio.h>

// Function to calculate and display simple interest
void calculateSimpleInterest(float principal, float rate, float time) {
    float interest = (principal * rate * time) / 100;
    printf("The simple interest is: %.2f\n", interest);
}

int main() {
    float principal, rate, time;

    // Input values
    printf("Enter the principal amount: ");
    scanf("%f", &principal);

    printf("Enter the rate of interest: ");
    scanf("%f", &rate);

    printf("Enter the time period in years: ");
    scanf("%f", &time);

    // Call the function to calculate and display simple interest
    calculateSimpleInterest(principal, rate, time);

    return 0;
}
```

7.Create a function that takes an integer and returns its reverse. Demonstrate how call by value affects the original number.

```
#include <stdio.h>

int reverse(int num) {
    int reversed = 0;
    while (num != 0) {
        reversed = reversed * 10 + num % 10;
        num /= 10;
    }
    return reversed;
}
```

```
int main() {
    int original = 12345;
    printf("Original number: %d\n", original);

    // Calling the reverse function
    int reversedNumber = reverse(original);

    printf("Reversed number: %d\n", reversedNumber);
    printf("Original number after reverse function call: %d\n", original); // Demonstrating call by value

    return 0;
}
```

WITHOUT RETURN TYPE

```
#include <stdio.h>

void reverse(int *num) {
    int reversed = 0;
    int original = *num; // To keep track of the original number if needed.

    while (*num != 0) {
        reversed = reversed * 10 + *num % 10;
        *num /= 10;
    }

    // Assign the reversed number back to the original variable
    *num = reversed;
}

int main() {
    int original = 12345;
    printf("Original number: %d\n", original);

    // Calling the reverse function
    reverse(&original);  // Pass the address of the original variable

    printf("Reversed number: %d\n", original);  // original is modified
    return 0;
}
```

8.Write a function to calculate the greatest common divisor (GCD) of two numbers passed by value

```
#include <stdio.h>

// Function to calculate GCD using Euclidean algorithm
int gcd(int a, int b) {
    // Keep dividing a by b until b becomes 0
    while (b != 0) {
        int temp = b;
        b = a % b;  // Get the remainder
        a = temp;   // Update a to be the previous b
    }
    return a;  // When b becomes 0, a contains the GCD
}

int main() {
    int num1, num2;

    // Input two numbers
    printf("Enter two numbers: ");
    scanf("%d %d", &num1, &num2);

    // Calculate and print the GCD
    printf("The GCD of %d and %d is: %d\n", num1, num2, gcd(num1, num2));

    return 0;
}
```

WITHOUT USING RETURN TYPE

```
#include <stdio.h>

// Function to calculate GCD using Euclidean algorithm without return type
void gcd(int a, int b, int *result) {
    while (b != 0) {
        int temp = b;
        b = a % b;  // Get the remainder
        a = temp;   // Update a to be the previous b
    }
    *result = a;  // Store the GCD in the memory location pointed to by result
}

int main() {
    int num1, num2, result;

    // Input two numbers
    printf("Enter two numbers: ");
    scanf("%d %d", &num1, &num2);

    // Call the gcd function
    gcd(num1, num2, &result);

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

    return 0;
}
```

9.Implement a function that computes the sum of the digits of a number passed as an argument.

```
#include <stdio.h>

int sum_of_digits(int number) {
    int sum = 0;

    // Handle negative numbers by taking the absolute value
    number = (number < 0) ? -number : number;

    // Sum the digits of the number
    while (number != 0) {
        sum += number % 10;  // Add the last digit to sum
        number /= 10;         // Remove the last digit
    }

    // Store the sum of digits
    int result = sum;

    // Return the result
    return result;
}

int main() {
    int num;

    // Input the number
    printf("Enter a number: ");
    scanf("%d", &num);

    // Store the result of sum_of_digits function
    int sum = sum_of_digits(num);

    // Output the sum of digits
    printf("Sum of digits: %d\n", sum);

    return 0;
}
WITHOUT RETURN TYPE
------------------------------
#include <stdio.h>

void sum_of_digits(int number, int *sum) {
    *sum = 0;  // Initialize sum to 0

    // Handle negative numbers by taking the absolute value
    number = (number < 0) ? -number : number;

    // Sum the digits of the number
    while (number != 0) {
        *sum += number % 10;  // Add the last digit to sum
        number /= 10;         // Remove the last digit
    }
}

int main() {
    int num;
    int sum = 0;

    // Input the number
    printf("Enter a number: ");
    scanf("%d", &num);

    // Call sum_of_digits function and pass the address of sum
    sum_of_digits(num, &sum);

    // Output the sum of digits
    printf("Sum of digits: %d\n", sum);
    return 0;
}
```

10.Write a program where a function checks if a given number is prime. Pass the number as an argument by value.

```
#include <stdio.h>

// Function to check if a number is prime
int isPrime(int num) {
    int result = 1;  // Assume the number is prime

    if (num <= 1) {
        result = 0;  // Numbers less than or equal to 1 are not prime
    } else {
        for (int i = 2; i * i <= num; i++) {  // Check divisibility up to the square root of num
            if (num % i == 0) {
                result = 0;  // num is divisible by i, so it is not prime
                break;
            }
        }
    }

    return result;  // Return the result (1 if prime, 0 if not)
}

int main() {
    int num;

    // Input the number from the user
    printf("Enter a number: ");
    scanf("%d", &num);

    // Call isPrime function and display result
    if (isPrime(num)) {
        printf("%d is a prime number.\n", num);
    } else {
        printf("%d is not a prime number.\n", num);
    }

    return 0;
}
```

WITHOUT RETURN TYPE

```
#include <stdio.h>

// Function to check if a number is prime
void isPrime(int num, int *result) {
    if (num <= 1) {
        *result = 0;  // Numbers less than or equal to 1 are not prime
    } else {
        *result = 1;  // Assume the number is prime
        for (int i = 2; i * i <= num; i++) {  // Check divisibility up to the square root of num
            if (num % i == 0) {
                *result = 0;  // num is divisible by i, so it is not prime
                break;
            }
        }
    }
}

int main() {
    int num, result;

    // Input the number from the user
    printf("Enter a number: ");
    scanf("%d", &num);

    // Call isPrime function and pass the address of result
    isPrime(num, &result);

    // Display the result
    if (result) {
        printf("%d is a prime number.\n", num);
    } else {
        printf("%d is not a prime number.\n", num);
    }

    return 0;
}
```

11.Create a function that checks whether a given number belongs to the Fibonacci sequence. Pass the number by value.

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

// Function to check if a number is a perfect square
int is_perfect_square(int n) {
    int sqrt_n = (int)sqrt(n);
    return (sqrt_n * sqrt_n == n);
}

// Function to check if a number is in the Fibonacci sequence
int is_fibonacci(int num) {
    // Store the value of the input number
    int value = num;

    // Check the two conditions for Fibonacci numbers
    if (is_perfect_square(5 * value * value + 4) || is_perfect_square(5 * value * value - 4)) {
        return value;  // Return the stored value if the number is a Fibonacci number
    } else {
        return -1;  // Return -1 if the number is not a Fibonacci number
    }
}

int main() {
    int num;
    printf("Enter a number to check if it’s a Fibonacci number: ");
    scanf("%d", &num);
    int result = is_fibonacci(num);
    if (result != -1) {
        printf("%d is a Fibonacci number.\n", result);
    } else {
        printf("%d is NOT a Fibonacci number.\n", num);
    }

    return 0;
}
```

WITHOUT RETURN TYPE

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

// Function to check if a number is a perfect square
int is_perfect_square(int n) {
    int sqrt_n = (int)sqrt(n);
    return (sqrt_n * sqrt_n == n);
}

// Function to check if a number is in the Fibonacci sequence
void is_fibonacci(int num) {
    // Check the two conditions for Fibonacci numbers
    if (is_perfect_square(5 * num * num + 4) || is_perfect_square(5 * num * num - 4)) {
        printf("%d is a Fibonacci number.\n", num);  // Print if it’s a Fibonacci number
    } else {
        printf("%d is NOT a Fibonacci number.\n", num);  // Print if it’s not a Fibonacci number
    }
}

int main() {
    int num;
    printf("Enter a number to check if it’s a Fibonacci number: ");
    scanf("%d", &num);

    is_fibonacci(num);  // Call the function to check and print the result

    return 0;
}
```

12.Write a function to calculate the roots of a quadratic equation ax2+bx+c=0ax^2 + bx + c = 0ax2+bx+c=0. Pass the coefficients a,b,a, b,a,b, and ccc as arguments.

```
#include <stdio.h>
#include <math.h>    // For sqrt() function

// Function to calculate roots of the quadratic equation and store the values in the provided pointers
int calculate_roots(double a, double b, double c, double *root1_real, double *root1_imag, double
*root2_real, double *root2_imag) {
    double discriminant = b * b - 4 * a * c;
    double realPart = -b / (2 * a);

    // Case 1: Two real roots
    if (discriminant > 0) {
        *root1_real = realPart + sqrt(discriminant) / (2 * a);
        *root2_real = realPart - sqrt(discriminant) / (2 * a);
        *root1_imag = *root2_imag = 0; // No imaginary part for real roots
        return 1; // Two real roots
    }
    // Case 2: One real root (repeated root)
    else if (discriminant == 0) {
        *root1_real = *root2_real = realPart;
        *root1_imag = *root2_imag = 0; // No imaginary part for repeated real root
        return 0; // One real root (repeated)
    }
    // Case 3: Two complex roots
    else {
        *root1_real = *root2_real = realPart;
        *root1_imag = sqrt(-discriminant) / (2 * a);
        *root2_imag = -(*root1_imag); // Complex conjugates
        return -1; // Two complex roots
    }
}

int main() {
    double a, b, c;
    double root1_real, root1_imag, root2_real, root2_imag;

    // Taking input for coefficients
    printf("Enter coefficients a, b, and c: ");
    scanf("%lf %lf %lf", &a, &b, &c);

    // Call the function to calculate and store roots
    int result = calculate_roots(a, b, c, &root1_real, &root1_imag, &root2_real, &root2_imag);

    // Display the roots
    if (result == 1) {
        printf("Root 1: %.2lf\n", root1_real);
        printf("Root 2: %.2lf\n", root2_real);
    } else if (result == 0) {
        printf("Root: %.2lf\n", root1_real); // Only one root for repeated real root
    } else {
        printf("Root 1: %.2lf + %.2lfi\n", root1_real, root1_imag);
        printf("Root 2: %.2lf - %.2lfi\n", root2_real, root2_imag);
    }

    return 0;
}
```

WITHOUT USING RETURN TYPE

```
#include <stdio.h>
#include <math.h>    // For sqrt() function

// Function to calculate roots of the quadratic equation and store them in passed pointers
void calculate_roots(double a, double b, double c, double *root1_real, double *root1_imag, double
*root2_real, double *root2_imag) {
    double discriminant = b * b - 4 * a * c;
    double realPart = -b / (2 * a);
    // Case 1: Two real roots
    if (discriminant > 0) {
        *root1_real = realPart + sqrt(discriminant) / (2 * a);
        *root2_real = realPart - sqrt(discriminant) / (2 * a);
        *root1_imag = *root2_imag = 0; // No imaginary part for real roots
    }
    // Case 2: One real root (repeated root)
    else if (discriminant == 0) {
        *root1_real = *root2_real = realPart;
        *root1_imag = *root2_imag = 0; // No imaginary part for repeated real root
    }
    // Case 3: Two complex roots
    else {
        *root1_real = *root2_real = realPart;
        *root1_imag = sqrt(-discriminant) / (2 * a);
        *root2_imag = -(*root1_imag); // Complex conjugates
    }
}

int main() {
    double a, b, c;
    double root1_real, root1_imag, root2_real, root2_imag;

    // Taking input for coefficients
    printf("Enter coefficients a, b, and c: ");
    scanf("%lf %lf %lf", &a, &b, &c);

    // Call the function to calculate and store roots
    calculate_roots(a, b, c, &root1_real, &root1_imag, &root2_real, &root2_imag);

    // Display the roots
    if (root1_imag == 0 && root2_imag == 0) {
        printf("Root 1: %.2lf\n", root1_real);
        printf("Root 2: %.2lf\n", root2_real);
    } else {
        printf("Root 1: %.2lf + %.2lfi\n", root1_real, root1_imag);
        printf("Root 2: %.2lf - %.2lfi\n", root2_real, root2_imag);
    }

    return 0;
}
```

13.Implement a function to convert a binary number (passed as an integer) into its decimal equivalent.

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

int binary_to_decimal(int binary) {
    int decimal = 0, remainder, i = 0;

    while (binary != 0) {
        remainder = binary % 10;  // Get the last digit (0 or 1)
        decimal += remainder * pow(2, i);  // Add the corresponding power of 2
        binary /= 10;  // Remove the last digit
        i++;  // Increment the power of 2
    }

    return decimal;
}

int main() {
    int binary_number;
    printf("Enter a binary number: ");
    scanf("%d", &binary_number);

    int decimal_number = binary_to_decimal(binary_number);
    printf("Decimal equivalent of binary %d is %d\n", binary_number, decimal_number);

    return 0;
}
```

WITHOUT USING RETURN TYPE

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

void binary_to_decimal(int binary, int *decimal) {
    *decimal = 0;
    int remainder, i = 0;

    while (binary != 0) {
        remainder = binary % 10;  // Get the last digit (0 or 1)
        *decimal += remainder * pow(2, i);  // Add the corresponding power of 2
        binary /= 10;  // Remove the last digit
        i++;  // Increment the power of 2
    }
}

int main() {
    int binary_number, decimal_number;
    printf("Enter a binary number: ");
    scanf("%d", &binary_number);

    binary_to_decimal(binary_number, &decimal_number);
    printf("Decimal equivalent of binary %d is %d\n", binary_number, decimal_number);

    return 0;
}
```

14.Write a program where a function computes the trace of a 2x2 matrix (sum of its diagonal elements). Pass the matrix elements individually as arguments

```
#include <stdio.h>

// Function to compute the trace of a 2x2 matrix and return it
int trace(int a, int b, int c, int d) {
    // Store the trace (sum of diagonal elements)
    int trace_value = a + d;
    return trace_value; // Return the trace value
}
```

```
int main() {
    int a, b, c, d;

    // Input the elements of the matrix
    printf("Enter the elements of the 2x2 matrix (a, b, c, d):\n");
    printf("a: ");
    scanf("%d", &a);
    printf("b: ");
    scanf("%d", &b);
    printf("c: ");
    scanf("%d", &c);
    printf("d: ");
    scanf("%d", &d);

    // Call the trace function to compute the trace
    int result = trace(a, b, c, d);

    // Print the computed trace
    printf("The trace of the matrix is: %d\n", result);

    return 0;
}
```

WITHOUT USING RETURN TYPE

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

// Function to compute the trace of a 2x2 matrix and store it in the provided pointer
void trace(int a, int b, int c, int d, int *result) {
    // Store the trace (sum of diagonal elements) into the variable pointed to by result
    *result = a + d;
}

int main() {
    int a, b, c, d, trace_value;

    // Input the elements of the matrix
    printf("Enter the elements of the 2x2 matrix (a, b, c, d):\n");
    printf("a: ");
    scanf("%d", &a);
    printf("b: ");
    scanf("%d", &b);
    printf("c: ");
    scanf("%d", &c);
    printf("d: ");
    scanf("%d", &d);

    // Call the trace function to compute the trace and store the result in trace_value
    trace(a, b, c, d, &trace_value);

    // Print the computed trace
    printf("The trace of the matrix is: %d\n", trace_value);

    return 0;
}
```

15.Create a function that checks whether a given number is a palindrome. Pass the number by value and return the result.

```
#include <stdio.h>

int isPalindrome(int num) {
    int originalNum = num;  // Store the original number
    int reversedNum = 0;
    int remainder;

    // Reverse the digits of the number
    while (num != 0) {
        remainder = num % 10;
        reversedNum = reversedNum * 10 + remainder;
        num /= 10;
    }

    // Check if the original number and the reversed number are the same
    if (originalNum == reversedNum) {
        return originalNum;  // Return the original number if it is a palindrome
    } else {
        return -1;  // Return -1 to indicate it’s not a palindrome
    }
}

int main() {
    int number;

    printf("Enter a number: ");
    scanf("%d", &number);

    int result = isPalindrome(number);

    if (result != -1) {
        printf("%d is a palindrome.\n", result);
    } else {
        printf("%d is not a palindrome.\n", number);
    }

    return 0;
}
```

WITHOUT USING RETURN TYPE

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

void isPalindrome(int num, int *result) {
    int originalNum = num;  // Store the original number
    int reversedNum = 0;
    int remainder;

    // Reverse the digits of the number
    while (num != 0) {
        remainder = num % 10;
        reversedNum = reversedNum * 10 + remainder;
        num /= 10;
    }

    // Check if the original number and the reversed number are the same
    if (originalNum == reversedNum) {
        *result = originalNum;  // Store the original number in result if it’s a palindrome
    } else {
        *result = -1;  // Store -1 in result to indicate it’s not a palindrome
    }
}

int main() {
    int number;
    int result;

    printf("Enter a number: ");
    scanf("%d", &number);

    // Call isPalindrome and pass the address of result
    isPalindrome(number, &result);

    if (result != -1) {
        printf("%d is a palindrome.\n", result);
    } else {
        printf("%d is not a palindrome.\n", number);
    }

    return 0;
}
```

SECOND SET OF PROBLEMS

1.A floating-point value representing the measurement and a character indicating the conversion type (e.g., ’C’ for cm-to-inches or ’I’ for inches-to-cm).

Output: The converted value.

Function:

float convert_units(float value, char type);

```
#include <stdio.h>

float convert_units(float value, char type) {
    // Check the type of conversion and perform the appropriate calculation
    if (type == ’C’) {
        // Convert from centimeters to inches
        return value * 0.393701;
    } else if (type == ’I’) {
        // Convert from inches to centimeters
        return value * 2.54;
    } else {
        // If the type is not recognized, return a special value (e.g., -1) to indicate an error
        return -1;
    }
}
```

```
int main() {
    // Example usage
    float value = 100.0;
    char type = ’C’; // Convert 100 cm to inches

    float result = convert_units(value, type);
    if (result != -1) {
        printf("Converted value: %.2f\n", result);
    } else {
        printf("Invalid conversion type.\n");
    }

    return 0;
}
```

2.Input: Two integers: the total length of the raw material and the desired length of each piece.
Output: The maximum number of pieces that can be cut and the leftover material.
Function:
int calculate_cuts(int material_length, int piece_length);

```
#include <stdio.h>

int calculate_cuts(int material_length, int piece_length) {
    if (piece_length == 0) {
        // To avoid division by zero
        printf("Error: Piece length cannot be zero.\n");
        return -1;
    }

    int num_pieces = material_length / piece_length;  // Number of pieces that can be cut
    int leftover = material_length % piece_length;    // Leftover material after cutting

    printf("Maximum number of pieces: %d\n", num_pieces);
    printf("Leftover material: %d\n", leftover);

    return num_pieces;  // You can choose to return the number of pieces or any other value.
}

int main() {
    int material_length, piece_length;

    // Input values
    printf("Enter the total material length: ");
    scanf("%d", &material_length);

    printf("Enter the desired piece length: ");
    scanf("%d", &piece_length);

    // Call the function
    calculate_cuts(material_length, piece_length);

    return 0;
}
```

3.Input: Two floating-point numbers: belt speed (m/s) and pulley diameter (m).
Output: The RPM of the machine.

Function:

```
float calculate_rpm(float belt_speed, float pulley_diameter);
```

```
#include <stdio.h>
// Define the value of Pi
#define PI 3.14159265358979323846

// Function to calculate RPM from belt speed and pulley diameter
float calculate_rpm(float belt_speed, float pulley_diameter) {
    // Calculate the circumference of the pulley
    float circumference = PI * pulley_diameter;

    // Calculate the RPM
    float rpm = (belt_speed / circumference) * 60.0;

    return rpm;
}

int main() {
    float belt_speed, pulley_diameter;

    // Example input
    printf("Enter belt speed (m/s): ");
    scanf("%f", &belt_speed);
    printf("Enter pulley diameter (m): ");
    scanf("%f", &pulley_diameter);

    // Calculate RPM
    float rpm = calculate_rpm(belt_speed, pulley_diameter);

    // Output the RPM
    printf("The RPM of the machine is: %.2f\n", rpm);

    return 0;
}
```

4.Input: Two integers: machine speed (units per hour) and efficiency (percentage).
Output: The effective production rate.
Function:

```
int calculate_production_rate(int speed, int efficiency);
```

```
#include <stdio.h>

int calculate_production_rate(int speed, int efficiency) {
    // Calculate the effective production rate by considering efficiency
    return (speed * efficiency) / 100;
}

int main() {
    int speed, efficiency;

    // Input speed and efficiency from the user
    printf("Enter machine speed (units per hour): ");
    scanf("%d", &speed);
    printf("Enter efficiency (percentage): ");
    scanf("%d", &efficiency);

    // Calculate and display the production rate
    int production_rate = calculate_production_rate(speed, efficiency);
    printf("The effective production rate is: %d units per hour\n", production_rate);

    return 0;
}
```

5.Input: Two integers: total material length and leftover material length.
Output: The amount of material wasted.
Function:
int calculate_wastage(int total_length, int leftover_length)

```
#include <stdio.h>

// Function to calculate the material wastage
int calculate_wastage(int total_length, int leftover_length) {
    // Wastage is the total length minus the leftover length
    return total_length - leftover_length;
}

int main() {
    // Example inputs
    int total_length = 100;
    int leftover_length = 20;

    // Calculate and display the wastage
    int wastage = calculate_wastage(total_length, leftover_length);
    printf("The amount of material wasted: %d\n", wastage);

    return 0;
}
```

6.Input: Three floating-point numbers: power rating (kW), operating hours, and cost per kWh.
Output: The total energy cost.
Function:
float calculate_energy_cost(float power_rating, float hours, float cost_per_kwh);

```
#include <stdio.h>

float calculate_energy_cost(float power_rating, float hours, float cost_per_kwh) {
    // Calculate the total energy cost
    return power_rating * hours * cost_per_kwh;
}

int main() {
    // Example input values
    float power_rating = 5.0;   // in kW
    float hours = 10.0;         // in hours
    float cost_per_kwh = 0.12;  // in cost per kWh (e.g., dollars)

    // Calculate energy cost
```

```c
    float total_cost = calculate_energy_cost(power_rating, hours, cost_per_kwh);

    // Print the result
    printf("Total energy cost: %.2f\n", total_cost);

    return 0;
}
```

7.Heat Generation in Machines

Input: Two floating-point numbers: power usage (Watts) and efficiency (%).

Output: Heat generated (Joules).

Function:

float calculate_heat(float power_usage, float efficiency);

```c
#include <stdio.h>

// Function to calculate the heat generated based on power usage and efficiency
float calculate_heat(float power_usage, float efficiency) {
    // Heat generated = Power usage * (1 - (efficiency / 100))
    return power_usage * (1 - (efficiency / 100));
}

int main() {
    float power_usage, efficiency, heat;

    // Taking input from the user
    printf("Enter power usage in Watts: ");
    scanf("%f", &power_usage);

    printf("Enter efficiency in percentage: ");
    scanf("%f", &efficiency);

    // Calculate heat generated
    heat = calculate_heat(power_usage, efficiency);

    // Output the result
    printf("The heat generated is: %.2f Joules\n", heat);

    return 0;
}
```

8.Tool Wear Rate Calculation

Input: A floating-point number for operating time (hours) and an integer for material type (e.g., 1 for metal, 2 for plastic).

Output: Wear rate (percentage).

Function:

float calculate_wear_rate(float time, int material_type);

```c
#include <stdio.h>

// Function to calculate wear rate based on operating time and material type
float calculate_wear_rate(float time, int material_type) {
    // Constants for material types
    float wear_rate_factor = 0.0;
    // Assign wear rate factor based on material type
    switch (material_type) {
        case 1: // Metal
            wear_rate_factor = 0.05; // Example constant for metal
            break;
        case 2: // Plastic
            wear_rate_factor = 0.10; // Example constant for plastic
            break;
        default:
            printf("Invalid material type\n");
            return -1.0; // Return -1 to indicate an error
    }

    // Calculate wear rate as a percentage based on time and material type
    float wear_rate = wear_rate_factor * time;

    // Ensure wear rate doesn’t exceed 100% (for realistic cases)
    if (wear_rate > 100.0) {
        wear_rate = 100.0;
    }

    return wear_rate;
}

int main() {
    float time;
    int material_type;

    // Example inputs
    printf("Enter operating time (in hours): ");
    scanf("%f", &time);
    printf("Enter material type (1 for Metal, 2 for Plastic): ");
    scanf("%d", &material_type);

    // Calculate the wear rate
    float wear_rate = calculate_wear_rate(time, material_type);

    if (wear_rate != -1.0) {
        printf("Wear rate: %.2f%%\n", wear_rate);
    }

    return 0;
}
```

9.Inventory Management
Input: Two integers: consumption rate (units/day) and lead time (days).
Output: Reorder quantity (units).
Function:
int calculate_reorder_quantity(int consumption_rate, int lead_time)

```
#include <stdio.h>

// Function to calculate reorder quantity
int calculate_reorder_quantity(int consumption_rate, int lead_time) {
    return consumption_rate * lead_time;
}

int main() {
    int consumption_rate, lead_time;

    // Input the consumption rate and lead time
    printf("Enter consumption rate (units/day): ");
    scanf("%d", &consumption_rate);

    printf("Enter lead time (days): ");
    scanf("%d", &lead_time);

    // Calculate and display the reorder quantity
    int reorder_quantity = calculate_reorder_quantity(consumption_rate, lead_time);
    printf("Reorder quantity (units): %d\n", reorder_quantity);

    return 0;
}
```

10.Quality Control: Defective Rate Analysis
Input: Two integers: number of defective items and total batch size.
Output: Defective rate (percentage).
Function:
float calculate_defective_rate(int defective_items, int batch_size);

```
#include <stdio.h>

// Function to calculate the defective rate
float calculate_defective_rate(int defective_items, int batch_size) {
    if (batch_size == 0) {
        // Handle the case where batch_size is 0 to avoid division by zero
        return 0.0;
    }

    // Calculate defective rate as a percentage
    float rate = ((float)defective_items / batch_size) * 100;
    return rate;
}

int main() {
    int defective_items, batch_size;

    // Input the number of defective items and batch size
    printf("Enter the number of defective items: ");
    scanf("%d", &defective_items);
    printf("Enter the total batch size: ");
    scanf("%d", &batch_size);

    // Calculate and display the defective rate
    float rate = calculate_defective_rate(defective_items, batch_size);
    printf("Defective rate: %.2f%%\n", rate);

    return 0;
}
```

11.Assembly Line Efficiency

Input: Two integers: output rate (units/hour) and downtime (minutes).
Output: Efficiency (percentage).
Function:

```
float calculate_efficiency(int output_rate, int downtime);
```

```
#include <stdio.h>

float calculate_efficiency(int output_rate, int downtime) {
    // Convert downtime from minutes to hours
    float downtime_hours = downtime / 60.0;

    // Calculate the total time in hours (assuming 1 hour is the total period)
    float total_time_hours = 1.0;

    // Calculate the actual output after downtime
    float actual_output = output_rate * (total_time_hours - downtime_hours);

    // Calculate the maximum possible output (if there were no downtime)
    float max_possible_output = output_rate * total_time_hours;

    // Calculate the efficiency as a percentage
    float efficiency = (actual_output / max_possible_output) * 100;

    return efficiency;
}

int main() {
    int output_rate = 50;  // Example output rate in units/hour
    int downtime = 15;     // Example downtime in minutes

    float efficiency = calculate_efficiency(output_rate, downtime);
    printf("The assembly line efficiency is: %.2f%%\n", efficiency);

    return 0;
}
```

12.Paint Coverage Estimation

Input: Two floating-point numbers: surface area (m^2) and paint coverage per liter (m^2/liter).
Output: Required paint (liters).
Function:

```
float calculate_paint(float area, float coverage);
```

```
#include <stdio.h>

// Function to calculate required paint (liters)
float calculate_paint(float area, float coverage) {
    // Calculate the required paint by dividing the area by coverage per liter
    return area / coverage;
}

int main() {
    float area, coverage, paint_needed;

    // Input surface area and paint coverage per liter
    printf("Enter surface area in square meters: ");
    scanf("%f", &area);

    printf("Enter paint coverage per liter in square meters: ");
    scanf("%f", &coverage);

    // Calculate the required paint
    paint_needed = calculate_paint(area, coverage);

    // Output the result
    printf("Required paint: %.2f liters\n", paint_needed);

    return 0;
}
```

13.Machine Maintenance Schedule

Input: Two integers: current usage (hours) and maintenance interval (hours).

Output: Hours remaining for maintenance.

Function:

```
int calculate_maintenance_schedule(int current_usage, int interval);
```

```
#include <stdio.h>

int calculate_maintenance_schedule(int current_usage, int interval) {
    // Calculate the remaining hours until the next maintenance
    int remaining_hours = interval - (current_usage % interval);

    // If remaining_hours equals the interval, it means maintenance is due now.
    if (remaining_hours == interval) {
        remaining_hours = 0;
    }

    return remaining_hours;
}

int main() {
    int current_usage, interval;

    // Take input from user for current usage and interval
    printf("Enter current usage (hours): ");
    scanf("%d", &current_usage);

    printf("Enter maintenance interval (hours): ");
    scanf("%d", &interval);

    // Calculate and output the remaining hours for maintenance
    int remaining = calculate_maintenance_schedule(current_usage, interval);
    printf("Hours remaining for next maintenance: %d\n", remaining);

    return 0;
}
```

14.Cycle Time Optimization

Input: Two integers: machine speed (units/hour) and number of operations per cycle.

Output: Optimal cycle time (seconds).

Function:
float calculate_cycle_time(int speed, int operations);

```
#include <stdio.h>

float calculate_cycle_time(int speed, int operations) {
    // Calculate the cycle time in seconds
    float cycle_time = (3600.0 * operations) / speed;
    return cycle_time;
}

int main() {
    int speed, operations;

    // Input machine speed (units per hour) and number of operations per cycle
    printf("Enter machine speed (units per hour): ");
    scanf("%d", &speed);

    printf("Enter number of operations per cycle: ");
    scanf("%d", &operations);

    // Calculate and display the optimal cycle time in seconds
    float optimal_cycle_time = calculate_cycle_time(speed, operations);
    printf("The optimal cycle time is: %.2f seconds\n", optimal_cycle_time);

    return 0;
}
```

THIRD SET OF PROGRAMS

1.Write a function that takes the original price of an item and a discount percentage as parameters. The function should return the discounted price without modifying the original price.
Function Prototype:
void calculateDiscount(float originalPrice, float discountPercentage);

```
#include <stdio.h>

void calculateDiscount(float originalPrice, float discountPercentage) {
    // Calculate the discount amount
    float discountAmount = (originalPrice * discountPercentage) / 100;

    // Calculate the discounted price
    float discountedPrice = originalPrice - discountAmount;

    // Print the discounted price
    printf("Original Price: $%.2f\n", originalPrice);
    printf("Discount Percentage: %.2f%%\n", discountPercentage);
    printf("Discounted Price: $%.2f\n", discountedPrice);
}

int main() {
    float originalPrice = 100.0;
    float discountPercentage = 20.0;

    // Call the calculateDiscount function
    calculateDiscount(originalPrice, discountPercentage);

    return 0;
}
```

2. Create a function that takes the current inventory count of a product and a quantity to add or remove. The function should return the new inventory count without changing the original count.
Function Prototype:
int updateInventory(int currentCount, int changeQuantity);

```
#include <stdio.h>

// Function prototype
int updateInventory(int currentCount, int changeQuantity);

int main() {
    int currentCount = 100;
    int changeQuantity = -20;

    // Update inventory without changing the original count
    int newCount = updateInventory(currentCount, changeQuantity);
    printf("Updated Inventory Count: %d\n", newCount);

    return 0;
}

// Function to update inventory without changing the original count
int updateInventory(int currentCount, int changeQuantity) {
    // Return the new count after adding/removing quantity
    return currentCount + changeQuantity;
}
```

3.Implement a function that accepts the price of an item and a sales tax rate. The function should return the total price after tax without altering the original price.
Function Prototype:
float calculateTotalPrice(float itemPrice, float taxRate);

```
#include <stdio.h>

// Function prototype
float calculateTotalPrice(float itemPrice, float taxRate);

int main() {
    float price, taxRate, totalPrice;

    // Example values
    price = 100.0;
    taxRate = 0.08; // 8% sales tax

    // Calling the function to calculate total price
    totalPrice = calculateTotalPrice(price, taxRate);

    // Printing the result
    printf("The total price after tax is: $%.2f\n", totalPrice);
    return 0;
}

// Function to calculate the total price after tax
float calculateTotalPrice(float itemPrice, float taxRate) {
    return itemPrice * (1 + taxRate); // Add tax to the original price
}
```

4.Design a function that takes the amount spent by a customer and returns the loyalty points earned based on a specific conversion rate (e.g., 1 point for every $10 spent). The original amount spent should remain unchanged.
Function Prototype:

```
int calculateLoyaltyPoints(float amountSpent);
```

```
#include <stdio.h>

int calculateLoyaltyPoints(float amountSpent) {
    // Define the conversion rate (1 point per $10)
    int conversionRate = 10;

    // Calculate the loyalty points
    int pointsEarned = (int)(amountSpent / conversionRate);

    // Return the points earned
    return pointsEarned;
}

int main() {
    // Test the function with an example
    float amountSpent = 123.45;  // Example amount spent
    int points = calculateLoyaltyPoints(amountSpent);

    // Output the result
    printf("Amount spent: $%.2f\n", amountSpent);
    printf("Loyalty points earned: %d\n", points);

    return 0;
}
```

5.Write a function that receives an array of item prices and the number of items. The function should return the total cost of the order without modifying the individual item prices.
Function Prototype:

```
float calculateOrderTotal(float prices[], int numberOfItems);
```

```
#include <stdio.h>

// Function prototype
float calculateOrderTotal(float prices[], int numberOfItems);

int main() {
    // Example array of item prices
    float prices[] = {10.99, 5.49, 3.25, 15.75};
    int numberOfItems = sizeof(prices) / sizeof(prices[0]); // Calculate number of items

    // Calculate and display the total cost
    float total = calculateOrderTotal(prices, numberOfItems);
    printf("The total cost of the order is: $%.2f\n", total);

    return 0;
}

// Function definition
float calculateOrderTotal(float prices[], int numberOfItems) {
    float total = 0.0;

    // Loop through the array to sum the prices
    for (int i = 0; i < numberOfItems; i++) {
        total += prices[i];
    }

    // Return the total cost
    return total;
}
```

6.Create a function that takes an item's price and a refund percentage as input. The function should return the refund amount without changing the original item's price.
Function Prototype:
float calculateRefund(float itemPrice, float refundPercentage);

```
#include <stdio.h>

// Function Prototype
float calculateRefund(float itemPrice, float refundPercentage);

int main() {
    float itemPrice, refundPercentage, refundAmount;

    // Input the item's price and the refund percentage
    printf("Enter item price: ");
    scanf("%f", &itemPrice);

    printf("Enter refund percentage: ");
    scanf("%f", &refundPercentage);

    // Calculate the refund amount
    refundAmount = calculateRefund(itemPrice, refundPercentage);

    // Output the refund amount
    printf("Refund amount: %.2f\n", refundAmount);

    return 0;
}

// Function Definition
float calculateRefund(float itemPrice, float refundPercentage) {
    // Calculate and return the refund amount
    return itemPrice * (refundPercentage / 100);
}
```

7.Implement a function that takes the weight of a package and calculates shipping costs based on weight

brackets (e.g., $5 for up to 5kg, $10 for 5-10kg). The original weight should remain unchanged.
Function Prototype:
float calculateShippingCost(float weight);

```
#include <stdio.h>

// Function prototype
float calculateShippingCost(float weight);

int main() {
   float weight;

   // Input the weight of the package
   printf("Enter the weight of the package (in kg): ");
   scanf("%f", &weight);

   // Calculate and display the shipping cost
   float cost = calculateShippingCost(weight);
   printf("The shipping cost for a package weighing %.2f kg is: $%.2f\n", weight, cost);

   return 0;
}

// Function to calculate shipping cost based on weight
float calculateShippingCost(float weight) {
   // Declare the shipping cost variable
   float cost;

   // Check the weight brackets and calculate the cost
   if (weight <= 5) {
      cost = 5.0;  // $5 for weight up to 5kg
   } else if (weight <= 10) {
      cost = 10.0; // $10 for weight between 5kg and 10kg
   } else if (weight <= 20) {
      cost = 15.0; // $15 for weight between 10kg and 20kg
   } else if (weight <= 50) {
      cost = 25.0; // $25 for weight between 20kg and 50kg
   } else {
      cost = 50.0; // $50 for weight over 50kg
   }

   // Return the calculated shipping cost
   return cost;
}
```

8.Design a function that converts an amount from one currency to another based on an exchange rate provided as input. The original amount should not be altered.
Function Prototype:
float convertCurrency(float amount, float exchangeRate);

```
#include <stdio.h>

// Function to convert currency
float convertCurrency(float amount, float exchangeRate) {
   return amount * exchangeRate;
}

int main() {
   float amount, exchangeRate, convertedAmount;

   // Input: Amount and exchange rate
   printf("Enter the amount in the original currency: ");
   scanf("%f", &amount);

   printf("Enter the exchange rate: ");
   scanf("%f", &exchangeRate);

   // Call the function to convert currency
   convertedAmount = convertCurrency(amount, exchangeRate);

   // Output: The converted amount
   printf("Converted amount: %.2f\n", convertedAmount);

   return 0;
}
```

9. Write a function that takes two prices from different vendors and returns the lower price without modifying either input price.
Function Prototype:
float findLowerPrice(float priceA, float priceB);

```
#include <stdio.h>

// Function prototype
float findLowerPrice(float priceA, float priceB);

int main() {
    float priceA = 19.99;
    float priceB = 25.50;

    printf("The lower price is: %.2f\n", findLowerPrice(priceA, priceB));
    return 0;
}

// Function definition
float findLowerPrice(float priceA, float priceB) {
    if (priceA < priceB) {
        return priceA;
    } else {
        return priceB;
    }
}
```

10. Create a function that checks if a customer is eligible for a senior citizen discount based on their age. The function should take age as input and return whether they qualify without changing the age value.
Function Prototype:
bool isEligibleForSeniorDiscount(int age);

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

```
int isEligibleForSeniorDiscount(int age) {
    // Check if age is 65 or older for senior citizen discount
    if (age >= 65) {
        return 1;  // Eligible for senior discount (return 1 for true)
    } else {
        return 0;  // Not eligible for senior discount (return 0 for false)
    }
}

int main() {
    int age;

    // Input age from user
    printf("Enter age: ");
    scanf("%d", &age);

    // Check eligibility for senior discount
    if (isEligibleForSeniorDiscount(age)) {
        printf("Eligible for senior citizen discount.\n");
    } else {
        printf("Not eligible for senior citizen discount.\n");
    }

    return 0;
}
```