1. Sum of Two Numbers

   Write a program that takes two integers as input and calculates their sum using a function. Pass the integers to the function using call by value.

   Without return type:

```
#include <stdio.h>
void sum2Elements(int, int); // function declaration

int main()
{
   int a = 20, b = 30;
   //call by value
   sum2Elements(a, b);
   printf("a = %d b = %d\n", a, b);
   return 0;
}

/*
Name: sum2Elements()
Return Type: void
Parameter:(data type of each parameter): int and int
Short description: it is used to add the 2 elements
*/

//function definition
void sum2Elements(int c, int d)
```

```
{
   int sum = 0;
   c = 30;
   d = 40;
   printf("c = %d d = %d\n", c, d);
   sum = c + d;
   printf("Sum = %d \n",sum);
}
```

With return type:

```
#include <stdio.h>
int sum2Elements(int c, int d); // function declaration

int main()
{
   int a = 20, b = 30, sumMain = 0;
   //call by value
   sumMain = sum2Elements(a, b);
   printf("a = %d b = %d\n", a, b);
   return 0;
}

/*
Name: sum2Elements()
Return Type: int
Parameter:(data type of each parameter): int and int
Short description: it is used to add the 2 elements
```

```
*/

//function definition
int sum2Elements(int c, int d)
{
   int sum = 0;
   c = 30;
   d = 40;
   printf("c = %d d = %d\n", c, d);
   sum = c + d;
   printf("Sum = %d \n",sum);
   return sum;
}
```

O/P:

```
c = 30 d = 40
Sum = 70
a = 20 b = 30
```

2. Swap Two Numbers

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

   Without return type:

```
#include <stdio.h>

void swap(int a, int b); // Function prototype
```

```
int main()
{
   int a = 10, b = 20;

   printf("Before swap: a = %d  b = %d\n", a, b);

   // Call the swap function by call by value
   swap(a, b);

   /* Original remains unchanged because the the function variables work
   on the copy of the data present in the variable and not on the original
   variable data */
   printf("1) After swap: a = %d  b = %d\n", a, b);

   return 0;
}

/*
Name: swap()
Return Type: void
Parameter:(data type of each parameter): int and int
Short description: it is used to swap 2 numbers
*/

// Function to swap two numbers
void swap(int a, int b)
{
```

```
    a = a + b;
    b = a - b;
    a = a - b;
    printf("2) After swap: a = %d  b = %d\n", a, b);
}
```

With return type:

```
#include <stdio.h>

int swap(int a, int b); // // Function prototype

int main()
{
    int a = 10, b = 20;
    printf("Before swap: a = %d, b = %d\n", a, b);

    // Call the swap function by call by value
    a = swap(a, b); // a gets the value of b
    b = swap(b, a); // b gets the value of a

    printf("After swap: a = %d, b = %d\n", a, b);
    return 0;
}

/*
Name: swap()
Return Type: int
```

Parameter:(data type of each parameter): int and int

Short description: it is used to swap 2 numbers

*/

```
// Function to swap two numbers
int swap(int a, int b)
{
   return b; // Return the value of another variable
}
```

O/P:

Before swap: a = 10 b = 20

After swap: a = 20 b = 10

3. Find Maximum of Two Numbers

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

Without return type:

```
#include <stdio.h>

void max2Numbers(int a, int b); // Function prototype

int main()
{
   int a = 20, b = 30;
```

```
    printf("Before the function call: a = %d  b = %d\n", a, b);

    // Call the max2Numbers function by call by value
    max2Numbers(a, b);

    /* Original remains unchanged because the the function variables work
    on the copy of the data present in the variable and not on the original
    variable data */
    printf("After the function call: a = %d  b = %d\n", a, b);

    return 0;
}

/*
Name: max2Numbers()
Return Type: void
Parameter:(data type of each parameter): int and int
Short description: it is used to find the larger of the 2 numbers
*/

// Function to find the larger of the two numbers
void max2Numbers(int a, int b)
{
    int maximum;
    if(a > b)
        maximum = a;
    else
        maximum = b;
```

```
    printf("Larger value: %d\n", maximum);
}
```

With return type:

```
#include <stdio.h>

int max2Numbers(int a, int b); // Function prototype

int main()
{
    int a = 20, b = 30;

    // Call the max2Numbers function by call by value
    int maximum = max2Numbers(a, b);

    printf("Larger value: %d\n", maximum);

    return 0;
}

/*
Name: max2Numbers()
Return Type: int
Parameter:(data type of each parameter): int and int
Short description: it is used to find the larger of the 2 numbers
*/

// Function to find the larger of the two numbers
```

```
int max2Numbers(int a, int b)
{
   if(a > b)
      return a;
   else
      return b;
}
```

O/P:

Before the function call: a = 20 b = 30

Larger value: 30

After the function call: a = 20 b = 30

4. Factorial Calculation

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

   Without return type:

```
#include <stdio.h>

void factorial(int num); // Function prototype

int main()
{
   int n = 5;

   printf("Before: n = %d\n", n);
```

```
    // Call the factorial function by call by value
    factorial(n);

    // Original number remains unaltered
    printf("After: n = %d\n", n);

    return 0;
}

/*
Name: factorial()
Return Type: void
Parameter:(data type of each parameter): int
Short description: it is used to find the factorial of a number
*/

// Function to find the factorial of a number
void factorial(int num)
{
    int fact = 1;
    for(int i = num; i >= 1; i--)
        fact = fact * i;
    printf("Factorial of %d is %d\n", num, fact);
}
```

With return type:

```
#include <stdio.h>

int factorial(int num); // Function prototype

int main()
{
   int n = 5;

   printf("Before: n = %d\n", n);

   // Call the factorial function by call by value
   int f = factorial(n);

   printf("Factorial of %d is %d\n", n, f);

   // Original number remains unaltered
   printf("After: n = %d\n", n);

   return 0;
}

/*
Name: factorial()
Return Type: int
Parameter:(data type of each parameter): int
Short description: it is used to find the factorial of a number
*/
```

```
// Function to find the factorial of a number
int factorial(int num)
{
   int fact = 1;
   for(int i = num; i >= 1; i--)
      fact = fact * i;
   return fact;
}
```

O/P:

Before: n = 5

Factorial of 5 is 120

After: n = 5

5. Check Even or Odd

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

   Without return type:

```
#include <stdio.h>

void evenOdd(int num); // Function prototype

int main()
{
   int n = 11;
```

```
    // Call the evenOdd function by call by value
    evenOdd(n);
    return 0;
}

/*
Name: evenOdd()
Return Type: void
Parameter:(data type of each parameter): int
Short description: it is used to find whether the number is even or odd
*/

// Function to find the whether the number is even or odd
void evenOdd(int num)
{
    if(num & 1)
        printf("%d is Odd\n", num);
    else
        printf("%d is Even\n", num);
}
```

With return type:

```
#include <stdio.h>

int evenOdd(int num); // Function prototype
```

```
int main()
{
   int n = 11;

   // Call the evenOdd function by call by value
   int result = evenOdd(n);

   if(result == 1)
      printf("%d is Odd\n", n);
   else
      printf("%d is Even\n", n);

   return 0;
}

/*
Name: evenOdd()
Return Type: int
Parameter:(data type of each parameter): int
Short description: it is used to find whether the number is even or odd
*/

// Function to find the whether the number is even or odd
int evenOdd(int num)
{
   if(num & 1)
      return 1;
```

```
    else
        return 0;
}
```

O/P:

11 is Odd

6. Calculate Simple Interest

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

Without return type:

```
#include <stdio.h>

void simpleInterest(float p, float r, float t); // Function prototype

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

    // Input the principal amount, rate of interest and time period
    printf("Enter the principal amount: ");
    scanf("%f", &principal);

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

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

    // Call the simpleInterest function
    simpleInterest(principal, rate, time);

    return 0;
}

/*
Name: simpleInterest()
Return Type: void
Parameter:(data type of each parameter): float, float and float
Short description: it is used to find calculate simple interest
*/

// Function to calculate simple interest
void simpleInterest(float p, float r, float t)
{
    float SI = (p * r * t) / 100;
    printf("Simple Interest = %.2f\n", SI);
}
```

With return type:

```
#include <stdio.h>

float simpleInterest(float p, float r, float t); // Function prototype
```

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

    // Input the principal amount, rate of interest and time period
    printf("Enter the principal amount: ");
    scanf("%f", &principal);

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

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

    // Call the simpleInterest function
    float SI = simpleInterest(principal, rate, time);
    printf("Simple Interest = %.2f\n", SI);

    return 0;
}

/*
Name: simpleInterest()
Return Type: float
Parameter:(data type of each parameter): float, float and float
Short description: it is used to find calculate simple interest
```

```
*/

// Function to calculate simple interest
float simpleInterest(float p, float r, float t)
{
   float SI = (p * r * t) / 100;
   return SI;
}
```

O/P:

```
Enter the principal amount: 5000
Enter the rate of interest: 5
Enter the time period: 1
Simple Interest = 250.00
```

7. Reverse a Number

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

   Without return type:

```
#include <stdio.h>

void reverseNumber(int num); // Function prototype

int main()
{
   int n;
```

```
    // Input the number to be reversed
    printf("Enter the number: ");
    scanf("%d", &n);

    // Before the function call
    printf("Before reversal: %d\n", n);

    // Call the reverseNmuber function by call by value
    reverseNumber(n);

    /*After the function call the call by value doesn't affect the original and
    it remains unchanged because the the function variables work on the
    copy of the data present in the variable and not on the original variable
    data */
    printf("After reversal: %d\n", n);

    return 0;
}

/*
Name: reverseNumber()
Return Type: void
Parameter:(data type of each parameter): int
Short description: it is used to reverse a number
*/

// Function to reverse a number
```

```
void reverseNumber(int num)
{
    int reverse;

    // Reverse the number
    for(reverse = 0; num != 0; num /= 10)
        reverse = reverse * 10 + num % 10;

    printf("The reversed number is: %d\n", reverse);
}
```

With return type:

```
#include <stdio.h>

int reverseNumber(int num); // Function prototype

int main()
{
    int n;

    // Input the number to be reversed
    printf("Enter the number: ");
    scanf("%d", &n);

    // Before the function call
    printf("Before reversal: %d\n", n);
```

```
    // Call the reverseNmuber function by call by value
    int r = reverseNumber(n);
    printf("The reversed number is: %d\n", r);

    /*After the function call the call by value doesn't affect the original and it remains unchanged because the the function variables work
     on the copy of the data present in the variable and not on the original variable data */
    printf("After reversal: %d\n", n);

    return 0;
}

/*
Name: reverseNumber()
Return Type: int
Parameter:(data type of each parameter): int
Short description: it is used to reverse a number
*/

// Function to reverse a number
int reverseNumber(int num)
{
    int reverse;

    // Reverse the number
    for(reverse = 0; num != 0; num /= 10)
        reverse = reverse * 10 + num % 10;
```

```
    return reverse;
}
```

O/P:

```
Enter the number: 45
Before reversal: 45
The reversed number is: 54
After reversal: 45
```

8. GCD of Two Numbers

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

   Without return type:

```
#include <stdio.h>

void gcd2Numbers(int a, int b); // Function prototype

int main()
{
    int a, b;

    // Input the two numbers
    printf("Enter the two numbers: ");
    scanf("%d %d", &a, &b);

    // Call the gdc2Numbers function by call by value
    gcd2Numbers(a, b);
```

```
    return 0;
}

/*
Name: gcd2Numbers()
Return Type: void
Parameter:(data type of each parameter): int and int
Short description: it is used to calculate GCD of 2 numbers
*/

// Function to calculate GCD of two numbers
void gcd2Numbers(int a, int b)
{
    while(a != b)
    {
        if(a > b)
            a = a - b;
        else
            b = b - a;
    }
    printf("GCD of the two numbers is: %d\n", b);
}
```

With return type:

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

```
int gcd2Numbers(int a, int b); // Function prototype

int main()
{
   int a, b;

   // Input the two numbers
   printf("Enter the two numbers: ");
   scanf("%d %d", &a, &b);

   // Call the gdc2Numbers function by call by value
   int gcd = gcd2Numbers(a, b);
   printf("GCD of the two numbers is: %d\n", gcd);

   return 0;
}

/*
Name: gcd2Numbers()
Return Type: int
Parameter:(data type of each parameter): int and int
Short description: it is used to calculate GCD of 2 numbers
*/

// Function to calculate GCD of two numbers
int gcd2Numbers(int a, int b)
{
```

```
    while(a != b)
    {
        if(a > b)
            a = a - b;
        else
            b = b - a;
    }
    return b;
}
```

O/P:

```
Enter the two numbers: 2 6
GCD of the two numbers is: 2
```

9. Sum of Digits

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

   Without return type:

```
#include <stdio.h>

void sumDigits(int num); // Function prototype

int main()
{
    int n;
```

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

    // Call the sumDigits function by call by value
    sumDigits(n);

    return 0;
}

/*
Name: sumDigits()
Return Type: void
Parameter:(data type of each parameter): int
Short description: it is used to compute the sum the digits of a number
*/

// Function to compute the sum the digits of a number
void sumDigits(int num)
{
    int sum;
    for(sum = 0; num; num /= 10)
        sum = sum + num % 10;

    printf("Sum of digits of a number: %d\n", sum);
}
```

With return type:

```
#include <stdio.h>

int sumDigits(int num); // Function prototype

int main()
{
   int n;

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

   // Call the sumDigits function by call by value
   int sum = sumDigits(n);
   printf("Sum of digits of a number: %d\n", sum);

   return 0;
}

/*
Name: sumDigits()
Return Type: int
Parameter:(data type of each parameter): int
Short description: it is used to compute the sum the digits of a number
*/
```

```
// Function to compute the sum the digits of a number
int sumDigits(int num)
{
   int sum;
   for(sum = 0; num; num /= 10)
      sum = sum + num % 10;
   return sum;
}
```

O/P:

Enter the number: 158

Sum of digits of a number: 14

10. Prime Number Check

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

Without return type:

```
#include <stdio.h>

void primeNumber(int num); // Function prototype

int main()
{
   int n;

   // Input the number
```

```
    printf("Enter the number: ");
    scanf("%d", &n);

    // Call the primeNumber function by call by value
    primeNumber(n);
    return 0;
}

/*
Name: primeNumber()
Return Type: void
Parameter:(data type of each parameter): int
Short description: it is used to check whether the number is prime or not
*/

// Function to check whether the number is prime or not
void primeNumber(int num)
{
    int i;
    for(i = 2; i < num; i++)
    {
        if(num % i == 0)
            break;
    }
    if(num == i)
        printf("%d is a prime number\n", num);
    else
```

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

With return type:

```
#include <stdio.h>

int primeNumber(int num); // Function prototype

int main()
{
   int n;

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

   // Call the primeNumber function by call by value
   int result = primeNumber(n);

   if(result == 1)
      printf("%d is a prime number\n", n);
   else
      printf("%d is not a prime number\n", n);

   return 0;
}

/*
```

```
Name: primeNumber()
Return Type: int
Parameter:(data type of each parameter): int
Short description: it is used to check whether the number is prime or not
*/

// Function to check whether the number is prime or not
int primeNumber(int num)
{
   int i;
   for(i = 2; i < num; i++)
   {
      if(num % i == 0)
         break;
   }
   if(num == i)
      return 1;
   else
      return 0;
}
```

O/P:

```
Enter the number: 11
11 is a prime number
```

11. Fibonacci Sequence Check

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

Without return type:

```
#include <stdio.h>

void fibonacciSequence(int num); // Function prototype

int main()
{
   int n;

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

   //Call the fibonacciSequence function by call by value
   fibonacciSequence(n);

   return 0;
}

/*
Name: fibonacciSequence()
Return Type: void
Parameter:(data type of each parameter): int
Short description: it is used to check whether the number belongs to fibonacci sequence
```

```
*/

// Function to check whether the number belongs to fibonacci sequence
void fibonacciSequence(int num)
{
   int a = 0, b = 1, i;

   while (b < num)
   {
      int c = a + b;
      a = b;
      b = c;
   }

   if (b == num)
      printf("%d belongs to the fibonacci sequence\n", num);
   else
      printf("%d does not belong to the fibonacci sequence\n", num);
}
```

With return type:

```
#include <stdio.h>

int fibonacciSequence(int num); // Function prototype

int main()
{
```

```
    int n;

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

    //Call the fibonacciSequence function by call by value
    int result = fibonacciSequence(n);
    if (result == n)
        printf("%d belongs to the fibonacci sequence\n", n);
    else
        printf("%d does not belong to the fibonacci sequence\n", n);

    return 0;
}

/*
Name: fibonacciSequence()
Return Type: int
Parameter:(data type of each parameter): int
Short description: it is used to check whether the number belongs to fibonacci sequence
*/

// Function to check whether the number belongs to fibonacci sequence
int fibonacciSequence(int num)
{
    int a = 0, b = 1, i;
```

```
    while (b < num)
    {
        int c = a + b;
        a = b;
        b = c;
    }

    if (b == num)
        return b;
    else
        return 0;
}
```

O/P:

Enter the number: 55

55 belongs to the fibonacci sequence

12. Quadratic Equation Solver

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 and c as arguments.

Without return type:

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

```
void quadraticRoots(int a, int b, int c); // Function prototype

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

   // Input the coefficients of a, b and c of the quadratic equation
   printf("Enter the coefficients: ");
   scanf("%d %d %d", &a, &b, &c);

   // Call the quadraticRoots function by call by value
   quadraticRoots(a, b, c);

   return 0;
}

/*
Name: quadraticRoots()
Return Type: void
Parameter:(data type of each parameter): int, int and int
Short description: it is used to calculate the roots of a quadratic equation
*/

// Function to calculate the roots of a quadratic equation
void quadraticRoots(int a, int b, int c)
{
   // Calculate the discriminant
```

```
    int discriminant = b * b - 4 * a * c;

    if (discriminant > 0)    // Two real and distinct roots
    {
        double root1 = (-b + sqrt(discriminant)) / (2 * a);
        double root2 = (-b - sqrt(discriminant)) / (2 * a);
        printf("The roots are real and distinct\n");
        printf("Root 1: %.2lf\n", root1);
        printf("Root 2: %.2lf\n", root2);
    }
    else if (discriminant == 0)      // One real and repeated root
    {
        double root = -b / (2 * a);
        printf("The root is real and repeated\n");
        printf("Root: %.2lf\n", root);
    }
    else     // Complex roots
    {
        double realPart = -b / (2 * a);
        double imaginaryPart = sqrt(-discriminant) / (2 * a);
        printf("The roots are complex\n");
        printf("Root 1: %.2lf + %.2lfi\n", realPart, imaginaryPart);
        printf("Root 2: %.2lf - %.2lfi\n", realPart, imaginaryPart);
    }
}
```

With return type:

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

int quadraticRoots(int a, int b, int c); // Function prototype

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

   // Input the coefficients of a, b and c of the quadratic equation
   printf("Enter the coefficients: ");
   scanf("%d %d %d", &a, &b, &c);

   // Call the quadraticRoots function by call by value
   int result = quadraticRoots(a, b, c);
   if(result == 1)
      printf("Enter the valid coefficient\n");

   return 0;
}

/*
Name: quadraticRoots()
Return Type: int
Parameter:(data type of each parameter): int, int and int
Short description: it is used to calculate the roots of a quadratic equation
*/
```

```
// Function to calculate the roots of a quadratic equation
int quadraticRoots(int a, int b, int c)
{
   if (a == 0)
   {
      printf("Invalid equation, the coefficient 'a' cannot be zero\n");
      return 1;
   }

   // Calculate the discriminant
   int discriminant = b * b - 4 * a * c;

   if (discriminant > 0)
   {
      // Two real and distinct roots
      double root1 = (-b + sqrt(discriminant)) / (2 * a);
      double root2 = (-b - sqrt(discriminant)) / (2 * a);
      printf("The roots are real and distinct\n");
      printf("Root 1: %.2lf\n", root1);
      printf("Root 2: %.2lf\n", root2);
      return 0;
   }
   else if (discriminant == 0)
   {
      // One real and repeated root
      double root = -b / (2 * a);
```

```
        printf("The root is real and repeated\n");
        printf("Root: %.2lf\n", root);
        return 0;
    }
    else
    {
        // Complex roots
        double realPart = -b / (2 * a);
        double imaginaryPart = sqrt(-discriminant) / (2 * a);
        printf("The roots are complex\n");
        printf("Root 1: %.2lf + %.2lfi\n", realPart, imaginaryPart);
        printf("Root 2: %.2lf - %.2lfi\n", realPart, imaginaryPart);
        return -1;
    }
}
```

O/P:

```
Enter the coefficients: 1 5 7
The roots are complex
Root 1: -2.00 + 0.87i
Root 2: -2.00 - 0.87i
```

13. Binary to Decimal Conversion

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

Without return type:

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

void binaryDecimal(int num); // Function prototype

int main()
{
   int n;

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

   // Call the binaryDecimal function by call by value
   binaryDecimal(n);

   return 0;
}

/*
Name: binaryDecimal()
Return Type: void
Parameter:(data type of each parameter): int
Short description: it is used to convert binary number into its decimal equivalent
*/

// Function to convert binary number into its decimal equivalent
```

```
void binaryDecimal(int num)
{
   int decimal = 0, base = 1, remainder;

   while (num > 0)
   {
      remainder = num % 10;
      decimal = decimal + remainder * base;
      num = num / 10;
      base = base * 2;
   }
   printf("Decimal equivalent: %d\n", decimal);
}
```

With return type:

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

int binaryDecimal(int num); // Function prototype

int main()
{
   int n;

   // Input the binary number
   printf("Enter the binary number: ");
   scanf("%d", &n);
```

```
    // Call the binaryDecimal function by call by value
    int result = binaryDecimal(n);
    printf("Decimal equivalent: %d\n", result);

    return 0;
}

/*
Name: binaryDecimal()
Return Type: int
Parameter:(data type of each parameter): int
Short description: it is used to convert binary number into its decimal equivalent
*/

// Function to convert binary number into its decimal equivalent
int binaryDecimal(int num)
{
    int decimal = 0, base = 1, remainder;

    while (num > 0)
    {
        remainder = num % 10;
        decimal = decimal + remainder * base;
        num = num / 10;
        base = base * 2;
    }
```

```
    return decimal;
}
```

O/P:

Enter the binary number: 1111

Decimal equivalent: 15

14. Matrix Trace Calculation

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.

Without return type:

```
#include <stdio.h>

void traceMatrix(int a, int b, int c, int d); //Function prototype

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

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

    // Call the traceMatrix function by call by value
    traceMatrix(a, b, c, d);
```

```
    return 0;
}

/*
Name: traceMatrix()
Return Type: void
Parameter:(data type of each parameter): int, int, int and int
Short description: it is used to compute the trace of a 2 x 2 matrix
*/

// Function to compute the trace of a 2 x 2 matrix
void traceMatrix(int a, int b, int c, int d)
{
    // The trace of a matrix is the sum of its diagonal elements
    int trace = a + d;

    printf("The trace of the matrix is: %d\n", trace);
}

With return type:
#include <stdio.h>

int traceMatrix(int a, int b, int c, int d); //Function prototype

int main()
{
```

```
    int a, b, c, d;

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

    // Call the traceMatrix function by call by value
    int result = traceMatrix(a, b, c, d);
    printf("The trace of the matrix is: %d\n", result);

    return 0;
}

/*
Name: traceMatrix()
Return Type: int
Parameter:(data type of each parameter): int, int, int and int
Short description: it is used to compute the trace of a 2 x 2 matrix
*/

// Function to compute the trace of a 2 x 2 matrix
int traceMatrix(int a, int b, int c, int d)
{
    // The trace of a matrix is the sum of its diagonal elements
    int trace = a + d;
    return trace;
}
```

O/P:

Enter the elements of the 2x2 matrix:

1 2

3 4

The trace of the matrix is: 5

15. Palindrome Number Check

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

Without return type:

```
#include <stdio.h>

void palindromeNumber(int num); // Function prototype

int main()
{
   int n;
   printf("Enter the number: ");
   scanf("%d", &n);

   //Call the palindromeNumber function by call by value
   palindromeNumber(n);

   return 0;
}
```

```
/*
Name: palindromeNumber()
Return Type: void
Parameter:(data type of each parameter): int
Short description: it is used to check whether the number is palindrome
*/

// Function to check whether the number is palindrome
void palindromeNumber(int num)
{
   int temp = num, reverse;
   for(reverse = 0; temp; temp /= 10)
      reverse = reverse * 10 + temp % 10;

   if(num == reverse)
      printf("%d is a palindrome number\n", num);
   else
      printf("%d is not a palindrome number\n", num);
}
```

With return type:

```
#include <stdio.h>

int palindromeNumber(int num); // Function prototype

int main()
```

```
{
    int n;
    printf("Enter the number: ");
    scanf("%d", &n);

    //Call the palindromeNumber function by call by value
    int result = palindromeNumber(n);
    if(result == 1)
        printf("%d is a palindrome number\n", n);
    else
        printf("%d is not a palindrome number\n", n);

    return 0;
}

/*
Name: palindromeNumber()
Return Type: int
Parameter:(data type of each parameter): int
Short description: it is used to check whether the number is palindrome
*/

// Function to check whether the number is palindrome
int palindromeNumber(int num)
{
    int temp = num, reverse;
    for(reverse = 0; temp; temp /= 10)
```

```
        reverse = reverse * 10 + temp % 10;

    if(num == reverse)
        return 1;
    else
        return 0;
}
```

O/P:

```
Enter the number: 12321
12321 is a palindrome number
```

1. Unit Conversion for Manufacturing Processes

- Input: 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); // Function prototype

int main()
```

```
{
    float value, converted_value;
    char type;

    // Input the value to conver and conversion type
    printf("Enter the value to convert: ");
    scanf("%f", &value);
    printf("Enter the conversion type ('C' for cm to inches, 'I' for inches to cm): ");
    scanf(" %c", &type);

    // Call the unitConversion function
    converted_value = convert_units(value, type);

    if (converted_value != -1)
        printf("Converted value: %.2f\n", converted_value);

    return 0;
}

/*
Name: convert_units()
Return Type: float
Parameter:(data type of each parameter): float and char
Short description: it is used to convert units
*/

// Function to convert units
```

```
float convert_units(float value, char type)
{
   float result;

   if (type == 'C')  // Convert cm to inches
      result = value * 0.393701;
   else if (type == 'I')  // Convert inches to cm
      result = value * 2.54;
   else  // Invalid conversion type
   {
      printf("Invalid conversion type\n");
      return -1;
   }
   return result;
}
```

O/P:

```
Enter the value to convert: 20
Enter the conversion type ('C' for cm to inches, 'I' for inches to cm): C
Converted value: 7.87

Enter the value to convert: 20
Enter the conversion type ('C' for cm to inches, 'I' for inches to cm): I
Converted value: 50.80
```

2. Cutting Material Optimization

- 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); // Function prototype


int main()
{
   int material_length, piece_length;


   // Input for the total length of raw material and desired piece length
   printf("Enter the total length of raw material: ");
   scanf("%d", &material_length);
   printf("Enter the desired length of each piece: ");
   scanf("%d", &piece_length);


   // Call the calculate_cuts function
   int result = calculate_cuts(material_length, piece_length);
   return 0;
}


/*
```

Name: calculate_cuts()

Return Type: int

Parameter:(data type of each parameter): int and int

Short description: it is used to calculate the maximum number of cuts and leftover material

*/

```
// Function to calculate the maximum number of cuts and leftover material
int calculate_cuts(int material_length, int piece_length)
{
    if (piece_length == 0)
    {
        printf("Piece length cannot be zero.\n");
        return -1;
    }

    int max_pieces = material_length / piece_length;
    int leftover = material_length % piece_length;

    printf("Maximum number of pieces: %d\n", max_pieces);
    printf("Leftover material: %d\n", leftover);
    return 0;
}
```

O/P:

```
Enter the total length of raw material: 200
Enter the desired length of each piece: 50
Maximum number of pieces: 4
```

Leftover material: 0

3. Machine Speed Calculation

- 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 PI 3.141

float calculate_rpm(float belt_speed, float pulley_diameter); // Function prototype

int main()
{
   float belt_speed, pulley_diameter, rpm;

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

   // Call the calculate_rpm function
```

```
    rpm = calculate_rpm(belt_speed, pulley_diameter);

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

    return 0;
}

/*
Name: calculate_rpm()
Return Type: float
Parameter:(data type of each parameter): float and float
Short description: it is used to calculate the RPM of the machine
*/

// Function to calculate the RPM of the machine
float calculate_rpm(float belt_speed, float pulley_diameter)
{
    // RPM formula: RPM = (belt_speed / (pi * pulley_diameter)) * 60
    float rpm = (belt_speed / (PI * pulley_diameter)) * 60;
    return rpm;
}
```

O/P:

```
Enter the belt speed (m/s): 20
Enter the pulley diameter (m): 0.8
The RPM of the machine is: 477.55
```

4. Production Rate Estimation

- 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); // Function prototype


int main()
{
   int speed, efficiency, production_rate;


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


   // Call the calculate_production_rate function
   production_rate = calculate_production_rate(speed, efficiency);


   printf("The effective production rate is: %d units per hour\n", production_rate);


   return 0;
```

```
}

/*
Name: calculate_production_rate()
Return Type: int
Parameter:(data type of each parameter): int and int
Short description: it is used to calculate the effective production rate
*/

// Function to calculate the effective production rate
int calculate_production_rate(int speed, int efficiency)
{
   int production_rate = (speed * efficiency) / 100;
   return production_rate;
}
```

O/P:

```
Enter the machine speed (units per hour): 120
Enter the efficiency (percentage): 50
The effective production rate is: 60 units per hour
```

5. Material Wastage Calculation

- 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>

int calculate_wastage(int total_length, int leftover_length); // Function prototype

int main()
{
   int total_length, leftover_length, wastage;

   // Input the total length of material and leftover length
   printf("Enter the total length of material: ");
   scanf("%d", &total_length);
   printf("Enter the leftover length of material: ");
   scanf("%d", &leftover_length);

   // Call the calculate_wastage function
   wastage = calculate_wastage(total_length, leftover_length);

   printf("The material wastage is: %d units\n", wastage);

   return 0;
}

/*
Name: calculate_wastage()
Return Type: int
Parameter:(data type of each parameter): int and int
```

Short description: it is used to calculate the material wastage

*/

```
// Function to calculate the material wastage
int calculate_wastage(int total_length, int leftover_length)
{
   int wastage = total_length - leftover_length;
   return wastage;
}
```

O/P:

Enter the total length of material: 50

Enter the leftover length of material: 10

The material wastage is: 40 units

6. Energy Cost Estimation

- 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);
// Function prototype
```

```
int main()
{
    float power_rating, hours, cost_per_kwh, total_cost;

    // Input the Power rating, operating hours, and cost per kWh
    printf("Enter the power rating of the device (kW): ");
    scanf("%f", &power_rating);
    printf("Enter the operating hours: ");
    scanf("%f", &hours);
    printf("Enter the cost per kWh: ");
    scanf("%f", &cost_per_kwh);

    // Call the calculate_energy_cost function
    total_cost = calculate_energy_cost(power_rating, hours, cost_per_kwh);

    printf("The total energy cost is: %.2f\n", total_cost);

    return 0;
}

/*
Name: calculate_energy_cost()
Return Type: float
Parameter:(data type of each parameter): float, float and float
Short description: it is used to calculate the total energy cost
*/
```

```
// Function to calculate the total energy cost
float calculate_energy_cost(float power_rating, float hours, float cost_per_kwh)
{
   float energy_cost = power_rating * hours * cost_per_kwh;
   return energy_cost;
}
```

O/P:

```
Enter the power rating of the device (kW): 50
Enter the operating hours: 2
Enter the cost per kWh: 1
The total energy cost is: 100.00
```

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);
```

```
#include <stdio.h>
float calculate_heat(float power_usage, float efficiency); // Function prototype

int main()
{
   float power_usage, efficiency, heat_generated;
```

```
    // Input the power usage in Watts and efficiency percentage
    printf("Enter the power usage (Watts): ");
    scanf("%f", &power_usage);
    printf("Enter the efficiency (percentage): ");
    scanf("%f", &efficiency);

    // Call the calculate_heat function to calculate heat generation
    heat_generated = calculate_heat(power_usage, efficiency);
    printf("The heat generated per second is: %.2f Joules\n", heat_generated);
    return 0;
}

/*
Name: calculate_heat()
Return Type: float
Parameter:(data type of each parameter): float and float
Short description: it is used to calculate the heat generation
*/

// Function to calculate heat generation per second
float calculate_heat(float power_usage, float efficiency)
{
    float heat_generated = power_usage * (1 - (efficiency / 100));
    return heat_generated;
}
```

O/P:

Enter the power usage (Watts): 60

Enter the efficiency (percentage): 50

The heat generated per second is: 30.00 Joules

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);
```

```
#include <stdio.h>

float calculate_wear_rate(float time, int material_type); // Function prototype

int main()
{
    float time, wear_rate;
    int material_type;

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

    // Call the calculate_wear_rate function to calculate wear rate
    wear_rate = calculate_wear_rate(time, material_type);
```

```
    if (wear_rate != -1)
        printf("The wear rate is: %.2f%%\n", wear_rate);
    else
        printf("Calculation failed due to invalid material type\n");

    return 0;
}

/*
Name: calculate_wear_rate()
Return Type: float
Parameter:(data type of each parameter): float and int
Short description: it is used to calculate the wear rate
*/

// Function to calculate the wear rate
float calculate_wear_rate(float time, int material_type)
{
    float wear_factor;
    if (material_type == 1) // Metal: 2% per hour
        wear_factor = 0.02;
    else if (material_type == 2) // Plastic: 1% per hour
        wear_factor = 0.01;
    else
    {
        printf("Invalid material type.\n");
        return -1;
```

```
    }

    float wear_rate = time * wear_factor;

    return wear_rate;
}
```

O/P:

Enter the operating time (hours): 20

Enter the material type (1 for Metal, 2 for Plastic): 1

The wear rate is: 0.40%

Enter the operating time (hours): 20

Enter the material type (1 for Metal, 2 for Plastic): 2

The wear rate is: 0.20%

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>

int calculate_reorder_quantity(int consumption_rate, int lead_time); // Function prototype
```

```
int main()
{
    int consumption_rate, lead_time, reorder_quantity;

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

    // Call the calculate_reorder_quantity function to calculate reorder quantity
    reorder_quantity = calculate_reorder_quantity(consumption_rate, lead_time);

    printf("The reorder quantity is: %d units\n", reorder_quantity);
    return 0;
}

/*
Name: calculate_reorder_quantity()
Return Type: int
Parameter:(data type of each parameter): int and int
Short description: it is used to calculate the reorder quantity
*/

// Function to calculate the reorder quantity
int calculate_reorder_quantity(int consumption_rate, int lead_time)
{
```

```
    int reorder_quantity = consumption_rate * lead_time;
    return reorder_quantity;
}
```

O/P:

Enter the consumption rate (units/day): 90

Enter the lead time (days): 5

The reorder quantity is: 450 units

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>

float calculate_defective_rate(int defective_items, int batch_size); //Function prototype

int main()
{
    int defective_items, batch_size;
    float defective_rate;

    // 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);

    // Call the calculate_defective_rate function to calculate defective rate
    defective_rate = calculate_defective_rate(defective_items, batch_size);

    if (defective_rate != -1)
        printf("The defective rate is: %.2f%%\n", defective_rate);

    return 0;
}

/*
Name: calculate_defective_rate()
Return Type: float
Parameter:(data type of each parameter): int and int
Short description: it is used to calculate the defective rate
*/

// Function to calculate the defective rate
float calculate_defective_rate(int defective_items, int batch_size)
{
    if (batch_size == 0)
    {
        printf("Batch size cannot be zero.\n");
        return -1;
    }
```

```
    float defective_rate = ((float)defective_items / batch_size) * 100;
    return defective_rate;
}
```

O/P:

Enter the number of defective items: 10

Enter the total batch size: 50

The defective rate is: 20.00%

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); // Function prototype

int main()
{
    int output_rate, downtime;
    float efficiency;

    // Input the output rate and downtime
    printf("Enter the output rate (units/hour): ");
    scanf("%d", &output_rate);
```

```
    printf("Enter the downtime (minutes): ");
    scanf("%d", &downtime);

    // Call the calculate_efficiency function to calculate efficiency
    efficiency = calculate_efficiency(output_rate, downtime);

    if (efficiency != -1)
        printf("Efficiency of the assembly line: %.2f%%\n", efficiency);

    return 0;
}

/*
Name: calculate_efficiency()
Return Type: float
Parameter:(data type of each parameter): int and int
Short description: it is used to calculate the assembly line efficiency
*/

// Function to calculate efficiency
float calculate_efficiency(int output_rate, int downtime)
{
    const int total_time = 60;

    if (downtime >= total_time)
    {
        printf("Downtime cannot exceed or equal the total time\n");
```

```
        return -1;
    }

    // Calculate actual operating time
    int actual_operating_time = total_time - downtime;

    // Calculate efficiency
    float efficiency = ((float)actual_operating_time / total_time) * 100;

    return efficiency;
}
```

O/P:

Enter the output rate (units/hour): 80

Enter the downtime (minutes): 40

Efficiency of the assembly line: 33.33%

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>

float calculate_paint(float area, float coverage); // Function prototype
```

```
int main()
{
    float area, coverage, required_paint;

    // Input the surface area and paint coverage
    printf("Enter the surface area to be painted (m²): ");
    scanf("%f", &area);
    printf("Enter the paint coverage per liter (m²/liter): ");
    scanf("%f", &coverage);

    // Call the function to calculate required paint
    required_paint = calculate_paint(area, coverage);

    if (required_paint != -1)
        printf("The required paint is: %.2f liters\n", required_paint);

    return 0;
}

/*
Name: calculate_paint()
Return Type: float
Parameter:(data type of each parameter): float and float
Short description: it is used to calculate the estimation of paint coverage
*/
```

```
// Function to calculate required paint
float calculate_paint(float area, float coverage)
{
   if (coverage <= 0)
   {
      printf("Paint coverage must be greater than zero.\n");
      return -1;
   }

   // Calculate required paint
   float required_paint = area / coverage;
   return required_paint;
}
```

O/P:

Enter the surface area to be painted (m^2): 30.5

Enter the paint coverage per liter (m^2/liter): 10

The required paint is: 3.05 liters

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); // Function prototype

int main()
{
   int current_usage, interval, remaining_hours;

   // Input the current usage and maintenance interval
   printf("Enter the current usage (hours): ");
   scanf("%d", &current_usage);
   printf("Enter the maintenance interval (hours): ");
   scanf("%d", &interval);

   // Call the function to calculate remaining hours for maintenance
   remaining_hours = calculate_maintenance_schedule(current_usage, interval);

   if (remaining_hours != -1)
   {
      if (remaining_hours == 0)
         printf("Maintenance is due now\n");
      else
        printf("Hours remaining for maintenance: %d hours\n", remaining_hours);
   }

   return 0;
}
```

```
/*
Name: calculate_maintenance_schedule()
Return Type: int
Parameter:(data type of each parameter): int and int
Short description: it is used to calculate the machine maintenance hours
*/

// Function to calculate hours remaining for maintenance
int calculate_maintenance_schedule(int current_usage, int interval)
{
    if (interval <= 0)
    {
        printf("Maintenance interval must be greater than zero.\n");
        return -1;
    }

    // Calculate hours remaining for maintenance
    int remaining_hours = interval - (current_usage % interval);

    // If the remaining hours equal the interval, it means maintenance is due now
    if (remaining_hours == interval)
        return 0;
    return remaining_hours;
}
```

O/P:

```
Enter the current usage (hours): 20
```

Enter the maintenance interval (hours): 5

Maintenance is due now

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); // Function prototype


int main()
{
   int speed, operations;


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


   // Calculate the optimal cycle time
   float cycle_time = calculate_cycle_time(speed, operations);
   if (cycle_time > 0)
      printf("Optimal cycle time: %.2f seconds\n", cycle_time);
```

```
    return 0;
}

/*
Name: calculate_cycle_time()
Return Type: float
Parameter:(data type of each parameter): int and int
Short description: it is used to calculate the cycle time optimization
*/

// Function to calculate cycle time
float calculate_cycle_time(int speed, int operations)
{
    if (speed <= 0 || operations <= 0)
    {
        printf("Speed and operations must be positive integers\n");
        return -1;
    }

    // Calculate cycle time in seconds
    float cycle_time = (3600.0 / speed) / operations;
    return cycle_time;
}
```

O/P:

```
Enter machine speed (units/hour): 150
Enter number of operations per cycle: 6
```

Optimal cycle time: 4.00 seconds

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); // Function prototype

int main()
{
    float originalPrice, discountPercentage;

    // Input the original price and discount percentage
    printf("Enter the original price of the item: ");
    scanf("%f", &originalPrice);
    printf("Enter the discount percentage: ");
    scanf("%f", &discountPercentage);

    // Call the function to calculate the discounted price
    calculateDiscount(originalPrice, discountPercentage);

    return 0;
}
```

```
/*
Name: calculateDiscount()
Return Type: void
Parameter:(data type of each parameter): float and float
Short description: it is used to calculate the discounted price
*/

// Function to calculate and display the discounted price
void calculateDiscount(float originalPrice, float discountPercentage)
{
    // Calculate the discount amount
    float discountAmount = (originalPrice * discountPercentage) / 100.0;

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

    printf("Original Price: %.2f\n", originalPrice);
    printf("Discount Percentage: %.2f%%\n", discountPercentage);
    printf("Discounted Price: %.2f\n", discountedPrice);
}
```

O/P:

```
Enter the original price of the item: 150
Enter the discount percentage: 20
Original Price: 150.00
Discount Percentage: 20.00%
```

Discounted Price: 120.00

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>

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

int main()
{
    int currentCount, changeQuantity;

    // Input the current inventory count and quantity to add/remove
    printf("Enter the current inventory count: ");
    scanf("%d", &currentCount);
    printf("Enter the quantity to add/remove (+ve for add, -ve for remove): ");
    scanf("%d", &changeQuantity);

    // Call the function to get the new inventory count
    int newCount = updateInventory(currentCount, changeQuantity);

    // Output the result
    printf("The new inventory count is: %d\n", newCount);

    return 0;
```

```
}

/*
Name: updateInventory()
Return Type: int
Parameter:(data type of each parameter): int and int
Short description: it is used to update the new inventory count
*/

// Function to update inventory and return the new count
int updateInventory(int currentCount, int changeQuantity)
{
   int newCount = currentCount + changeQuantity;
   return newCount;
}
```

O/P:

```
Enter the current inventory count: 100
Enter the quantity to add/remove (+ve for add, -ve for remove): +20
The new inventory count is: 120

Enter the current inventory count: 150
Enter the quantity to add/remove (+ve for add, -ve for remove): -10
The new inventory count is: 140
```

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>

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

int main()
{
    float itemPrice, taxRate;

    // Input the item price and sales tax rate
    printf("Enter the item price: ");
    scanf("%f", &itemPrice);
    printf("Enter the sales tax rate: ");
    scanf("%f", &taxRate);

    // Call the function to calculate the total price after tax
    float totalPrice = calculateTotalPrice(itemPrice, taxRate);

    printf("The total price after tax is: %.2f\n", totalPrice);

    return 0;
}

/*
Name: calculateTotalPrice()
```

Return Type: float

Parameter:(data type of each parameter): float and float

Short description: it is used to calculate the total price aftertax

*/

```
// Function to calculate total price after tax
float calculateTotalPrice(float itemPrice, float taxRate)
{
   // Calculate the tax amount
   float taxAmount = (itemPrice * taxRate) / 100.0;

   // Calculate the total price including tax
   float totalPrice = itemPrice + taxAmount;

   return totalPrice;
}
```

O/P:

```
Enter the item price: 1000
Enter the sales tax rate: 25
The total price after tax is: 1250.00
```

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); // Function prototype

int main()
{
    float amountSpent;

    // Input the amount spent by the customer
    printf("Enter the amount spent by the customer: ");
    scanf("%f", &amountSpent);

    // Call the function to calculate loyalty points
    int loyaltyPoints = calculateLoyaltyPoints(amountSpent);

    printf("The customer earned %d loyalty points\n", loyaltyPoints);

    return 0;
}

/*
Name: calculateLoyaltyPoints()
Return Type: int
Parameter:(data type of each parameter): float
Short description: it is used to calculate the loyalty points earned based on a specific conversion rate
*/
```

```
// Function to calculate loyalty points based on amount spent
int calculateLoyaltyPoints(float amountSpent)
{
   // Define the conversion rate (1 point for every $10 spent)
   const float conversionRate = 10.0;

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

   return loyaltyPoints;
}
```

O/P:

Enter the amount spent by the customer: 1500

The customer earned 150 loyalty points

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>

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

```
int main()
{
    int numberOfItems;

    // Input the number of items in the order
    printf("Enter the number of items: ");
    scanf("%d", &numberOfItems);

    float prices[numberOfItems];

    // Input the prices of the items
    printf("Enter the prices of the items\n");
    for (int i = 0; i < numberOfItems; i++) {
        printf("Price of item %d: ", i + 1);
        scanf("%f", &prices[i]);
    }

    // Call the function to calculate the total cost
    float totalCost = calculateOrderTotal(prices, numberOfItems);

    printf("The total cost of the order is: %.2f\n", totalCost);

    return 0;
}

/*
```

Name: calculateOrderTotal()

Return Type: float

Parameter:(data type of each parameter): float and int

Short description: it is used to calculate the total cost of the order

*/

```
// Function to calculate the total cost of the order
float calculateOrderTotal(float prices[], int numberOfItems)
{
   float total = 0.0;

   for (int i = 0; i < numberOfItems; i++) {
      total += prices[i];
   }

   return total;
}
```

O/P:

```
Enter the number of items: 5
Enter the prices of the items:
Price of item 1: 100
Price of item 2: 50
Price of item 3: 75
Price of item 4: 83
Price of item 5: 45
The total cost of the order is: 353.00
```

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>

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

int main()
{
   float itemPrice, refundPercentage;

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

   // Call the function to calculate the refund amount
   float refundAmount = calculateRefund(itemPrice, refundPercentage);

   printf("The refund amount is: %.2f\n", refundAmount)
   return 0;
}
```

```
/*
Name: calculateRefund()
Return Type: float
Parameter:(data type of each parameter): float and float
Short description: it is used to calculate the refund amount
*/

// Function to calculate refund amount
float calculateRefund(float itemPrice, float refundPercentage)
{
   float refundAmount = (itemPrice * refundPercentage) / 100.0;
   return refundAmount;
}
```

O/P:

```
Enter the item price: 2000
Enter the refund percentage: 20
The refund amount is: 400.00
```

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>
```

```
float calculateShippingCost(float weight); // Function prototype

int main()
{
   float weight;

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

   // Call the function to calculate the shipping cost
   float shippingCost = calculateShippingCost(weight);

   printf("The shipping cost for a %.2f kg package is: %.2f\n", weight, shippingCost);
   return 0;
}

/*
Name: calculateShippingCost()
Return Type: float
Parameter:(data type of each parameter): float
Short description: it is used to calculate the shipping cost based on weight
*/

// Function to calculate shipping cost based on weight
float calculateShippingCost(float weight)
{
```

```
    float shippingCost = 0.0;

    // Determine shipping cost based on weight brackets
    if (weight <= 5)    // $5 for up to 5kg
        shippingCost = 5.00;
    else if (weight <= 10)   // $10 for 5-10kg
        shippingCost = 10.00;
    else if (weight <= 20)  // $15 for 10-20kg
        shippingCost = 15.00;
    else    // $20 for over 20kg
        shippingCost = 20.00;

    return shippingCost;
}
```

O/P:

```
Enter the weight of the package (in kg): 8.5
The shipping cost for a 8.50 kg package is: 10.00
```

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>

float convertCurrency(float amount, float exchangeRate); // Function prototype
```

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

    // Input the amount to be converted and exchange rate
    printf("Enter the amount to convert: ");
    scanf("%f", &amount);
    printf("Enter the exchange rate: ");
    scanf("%f", &exchangeRate);

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

    printf("The converted amount is: %.2f\n", convertedAmount);
    return 0;
}

/*
Name: convertCurrency()
Return Type: float
Parameter:(data type of each parameter): float and float
Short description: it is used to convert currency on exchange rate
*/

// Function to convert currency based on exchange rate
float convertCurrency(float amount, float exchangeRate)
```

```
{
   float convertedAmount = amount * exchangeRate;
   return convertedAmount;
}
```

O/P:

Enter the amount to convert: 100

Enter the exchange rate: 8

The converted amount is: 800.00

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>

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

int main()
{
   float priceA, priceB;

   // Input the prices from two different vendors
   printf("Enter the price from vendor A: ");
   scanf("%f", &priceA);
   printf("Enter the price from vendor B: ");
```

```
    scanf("%f", &priceB);

    // Call the function to find the lower price
    float lowerPrice = findLowerPrice(priceA, priceB);

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

/*
Name: findLowerPrice()
Return Type: float
Parameter:(data type of each parameter): float and float
Short description: it is used to find lower price from different vendors
*/

// Function to find and return the lower price
float findLowerPrice(float priceA, float priceB)
{
    if (priceA < priceB)
        return priceA;
    else
        return priceB;
}
```

O/P:

```
        Enter the price from vendor A: 100
```

Enter the price from vendor B: 120

The lower price is: 100.00

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>
#include <stdbool.h>

bool isEligibleForSeniorDiscount(int age); // Function prototype

int main()
{
    int age;

    // Input the customer's age
    printf("Enter the customer's age: ");
    scanf("%d", &age);

    // Call the function to check eligibility for senior discount
    bool eligible = isEligibleForSeniorDiscount(age);

    if (eligible)
        printf("The customer is eligible for a senior citizen discount\n");
```

```
    else
        printf("The customer is not eligible for a senior citizen discount\n");
    return 0;
}

/*
Name: isEligibleForSeniorDiscount()
Return Type: bool
Parameter:(data type of each parameter): int
Short description: it is used to check whether the customer is eligible for a senior citizen discount based on age
*/

// Function to check if the customer is eligible for a senior citizen discount
bool isEligibleForSeniorDiscount(int age)
{
    // Senior citizen age is typically considered 65 or older
    if (age >= 65)
        return true;
    else
        return false;
}
```

O/P:

```
Enter the customer's age: 75
The customer is eligible for a senior citizen discount
```