Recursive functions

1. Factorial Calculation: Write a recursive function to calculate the factorial of a given non-negative integer n.

   Without pointers:

```
#include <stdio.h>

int fact(int n);

int main()
{
   int num, f;
   printf("Enter the number: ");
   scanf("%d", &num);
   f = fact(num);
   printf("Factorial of %d = %d\n", num, f);
   return 0;
}

int fact(int n)
{
   if(n)
      return n * fact(n - 1);
   else
      return 1;
}
```

With pointers:

```
#include <stdio.h>

int fact(int *n);

int main()
{
   int num, f;
   printf("Enter the number: ");
   scanf("%d", &num);
   f = fact(&num);
   printf("Factorial = %d\n", f);
   return 0;
}

int fact(int *n)
{
   if (*n > 0)
   {
      int f1 = *n;
      (*n)--;
      return f1 * fact(n);
   }
   else
      return 1;
}
```

2. Fibonacci Series: Create a recursive function to find the nth term of the Fibonacci series.

Without pointers:

```
#include <stdio.h>

int fibonacci(int n);

int main()
{
   int n, f;
   printf("Enter the nth term value: ");
   scanf("%d", &n);
   if (n < 0)
   {
      printf("Invalid input\n");
      return 1;
   }
   f = fibonacci(n);
   printf("The %dth term in the fibonacci series is: %d\n", n, f);
   return 0;
}

int fibonacci(int n)
{
   if (n == 0)
      return 0;
   if (n == 1)
      return 1;
```

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

With pointers:

```
#include <stdio.h>

int fibonacci(int *n);

int main()
{
    int n, f;
    printf("Enter the nth term value: ");
    scanf("%d", &n);
    if (n < 0)
    {
        printf("Invalid input\n");
        return 1;
    }
    f = fibonacci(&n);
    printf("The %dth term in the Fibonacci series is: %d\n", n, f);
    return 0;
}

int fibonacci(int *n)
{
    if (*n == 0)
        return 0;
    if (*n == 1)
        return 1;
```

```
    int t = *n;
    int p = t - 1;
    int q = t - 2;
    return fibonacci(&p) + fibonacci(&q);
}
```

3. Sum of Digits: Implement a recursive function to calculate the sum of the digits of a given positive integer.

Without pointers:

```
#include <stdio.h>

int sumOfDigits(int n);

int main()
{
    int num, sum;
    printf("Enter the number: ");
    scanf("%d", &num);
    if (num < 0)
    {
        printf("Invalid input\n");
        return 1;
    }
    sum = sumOfDigits(num);
    printf("Sum of the digits = %d\n", sum);
    return 0;
}
```

```
int sumOfDigits(int n)
{
   if (n == 0)
      return 0;
   return (n % 10) + sumOfDigits(n / 10);
}
```

With pointers:

```
#include <stdio.h>

int sumOfDigits(int *n);

int main()
{
   int num, sum;
   printf("Enter the number: ");
   scanf("%d", &num);
   if (num < 0)
   {
      printf("Invalid input\n");
      return 1;
   }
   sum = sumOfDigits(&num);
   printf("Sum of the digits = %d\n", sum);
   return 0;
}

int sumOfDigits(int *n)
{
```

```
    if (*n == 0)
        return 0;
    int n1 = *n % 10;
    *n = *n / 10;
    return n1 + sumOfDigits(n);
}
```

4. Reverse a String: Write a recursive function to reverse a string.

Without pointers:

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

void reverseString(char a[], int s, int e);

int main()
{
    char str[50];
    printf("Enter a string: ");
    scanf("%s", str);
    int l = strlen(str);
    reverseString(str, 0, l - 1);
    printf("Reversed string: %s\n", str);
    return 0;
}

void reverseString(char a[], int s, int e)
{
```

```
    if (s >= e)
        return;
    char t = a[s];
    a[s] = a[e];
    a[e] = t;
    reverseString(a, s + 1, e - 1);
}
```

With pointers:

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

void reverseString(char *a, int s, int e);

int main()
{
    char str[50];
    printf("Enter a string: ");
    scanf("%s", str);

    int l = strlen(str);
    reverseString(str, 0, l - 1);
    printf("Reversed string: %s\n", str);

    return 0;
}

void reverseString(char *a, int s, int e)
{
```

```
    if (s >= e)
        return;
    char t = *(a + s);
    *(a + s) = *(a + e);
    *(a + e) = t;
    reverseString(a, s + 1, e - 1);
}
```

5. Power Calculation: Develop a recursive function to calculate the power of a number x raised to n Greatest Common Divisor (GCD): Create a recursive function to find the GCD of two given integers using the Euclidean algorithm.

Without pointers:

```
#include <stdio.h>

int power(int x, int n);
int gcd(int a, int b);

int main()
{
    int x, n, a, b;

    // Power Calculation
    printf("Enter the base number x: ");
    scanf("%d", &x);
    printf("Enter the exponent n: ");
    scanf("%d", &n);
```

```
    printf("%d raised to the power of %d is: %d\n", x, n, power(x, n));

    // GCD Calculation
    printf("\nEnter two integers to calculate GCD: ");
    scanf("%d %d", &a, &b);
    printf("The GCD of %d and %d is: %d\n", a, b, gcd(a, b));
    return 0;
}

int power(int x, int n)
{
    if (n == 0)
        return 1;
    return x * power(x, n - 1);
}

int gcd(int a, int b)
{
    if (b == 0)
        return a;
    return gcd(b, a % b);
}
```

With pointers:

```
#include <stdio.h>

int power(int *x, int *n);
int gcd(int *a, int *b);
int main()
```

```
{
    int x, n, a, b;

    // Power Calculation
    printf("Enter the base number x: ");
    scanf("%d", &x);
    printf("Enter the exponent n: ");
    scanf("%d", &n);
    printf("%d raised to the power of %d is: %d\n", x, n, power(&x, &n));

    // GCD Calculation
    printf("\nEnter two integers to calculate GCD: ");
    scanf("%d %d", &a, &b);
    printf("GCD = %d\n", gcd(&a, &b));
    return 0;
}

int power(int *x, int *n)
{
    if (*n == 0)
        return 1;
    return *x * power(x, n - 1);
}

int gcd(int *a, int *b)
{
    while (*b != 0)
    {
        int temp = *b;
```

```
        *b = *a % *b;
        *a = temp;
    }
    return *a;
}
```

6. Count Occurrences of a Character: Develop a recursive function to count the number of times a specific character appears in a string.

Without pointers:

```
#include <stdio.h>

int countCharacter(char str[], char ch);

int main()
{
    char str[100], ch;
    printf("Enter a string: ");
    scanf("%s", str);
    printf("Enter the character to count: ");
    scanf(" %c", &ch);
    int count = countCharacter(str, ch);
    printf("The character '%c' appears %d times in the string\n", ch, count);
    return 0;
}

int countCharacter(char str[], char ch)
{
```

```
    if (str[0] == '\0')
        return 0;
    if (str[0] == ch)
        return 1 + countCharacter(str + 1, ch);
    return countCharacter(str + 1, ch);
}
```

With pointers:

```
#include <stdio.h>

int countCharacter(char *str, char ch);

int main()
{
    char str[100], ch;
    printf("Enter a string: ");
    scanf("%s", str);
    printf("Enter the character to count: ");
    scanf(" %c", &ch);
    int count = countCharacter(str, ch);
    printf("The character '%c' appears %d times in the string\n", ch, count);
    return 0;
}

int countCharacter(char *str, char ch)
{
    if (*str == '\0')
        return 0;
    if (*str == ch)
```

```
        return 1 + countCharacter(str + 1, ch);
    return countCharacter(str + 1, ch);
}
```

7. Palindrome Check: Create a recursive function to check if a given string is a palindrome.

Without pointers:

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

int palindrome(char str[], int s, int e);

int main()
{
    char str[100];
    printf("Enter a string: ");
    scanf("%s", str);
    int l = strlen(str);
    if (palindrome(str, 0, l - 1))
        printf("String is a palindrome\n");
    else
        printf("String is not a palindrome\n");
    return 0;
}

int palindrome(char str[], int s, int e)
{
```

```
    if (s >= e)
        return 1;
    if (str[s] != str[e])
        return 0;
    return palindrome(str, s + 1, e - 1);
}
```

With pointers:

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

int palindrome(char *str, int s, int e);

int main()
{
    char str[100];
    printf("Enter a string: ");
    scanf("%s", str);
    int l = strlen(str);
    if (palindrome(str, 0, l - 1))
        printf("String is a palindrome\n");
    else
        printf("String is not a palindrome\n");
    return 0;
}

int palindrome(char *str, int s, int e)
{
    if (s >= e)
```

```
        return 1;
    if (*(str + s) != *(str + e))
        return 0;
    return palindrome(str, s + 1, e - 1);
}
```

8. String Length: Write a recursive function to calculate the length of a given string without using any library functions.

Without pointers:

```
#include <stdio.h>

int stringLength(char str[], int i);

int main()
{
    char str[100];
    printf("Enter a string: ");
    scanf("%s", str);
    int l = stringLength(str, 0);
    printf("The length of the string is: %d\n", l);
    return 0;
}

int stringLength(char str[], int i)
{
    if (str[i] == '\0')
        return 0;
```

```
    return 1 + stringLength(str, i + 1);
}
```

With pointers:

```
#include <stdio.h>

int stringLength(char *str);

int main()
{
    char str[100];
    printf("Enter a string: ");
    scanf("%s", str);
    int l = stringLength(str);
    printf("The length of the string is: %d\n", l);
    return 0;
}

int stringLength(char *str)
{
    if (*str == '\0')
        return 0;
    return 1 + stringLength(str + 1);
}
```

9. Check for Prime Number: Implement a recursive function to check if a given number is a prime number.

Without pointers:

```
#include <stdio.h>

int prime(int n, int i);

int main()
{
   int num;
   printf("Enter a number: ");
   scanf("%d", &num);
   if (num <= 1)
      printf("%d is not a prime number\n", num);
   else
   {
      if (prime(num, 2))
         printf("%d is a prime number\n", num);
      else
         printf("%d is not a prime number\n", num);
   }
   return 0;
}

int prime(int n, int i)
{
   if (i * i > n)
      return 1;
   if (n % i == 0)
      return 0;
   return prime(n, i + 1);
```

```
}
```

With pointers:

```
#include <stdio.h>

int prime(int *n, int *i);

int main()
{
   int num;
   printf("Enter a number: ");
   scanf("%d", &num);
   if (num <= 1)
      printf("%d is not a prime number\n", num);
   else
   {
      if (prime(&num, &(int){2}))
         printf("%d is a prime number\n", num);
      else
         printf("%d is not a prime number\n", num);
   }
   return 0;
}

int prime(int *n, int *i)
{
   if (*i * *i > *n)
      return 1;
   if (*n % *i == 0)
```

```
        return 0;
    return prime(n, i + 1);
}
```

10. Print Numbers in Reverse: Create a recursive function to print the numbers from n down to 1 in reverse order.

Without pointers:

```
#include <stdio.h>

void printReverse(int n);

int main()
{
    int num;
    printf("Enter a number: ");
    scanf("%d", &num);
    printf("Numbers in reverse order from %d  to 1:\n", num);
    printReverse(num);
    return 0;
}

void printReverse(int n)
{
    if (n <= 0)
        return;
    printf("%d ", n);
    printReverse(n - 1);
```

```
}
```

With pointers:

```
#include <stdio.h>

void printReverse(int *n);

int main()
{
   int num;
   printf("Enter a number: ");
   scanf("%d", &num);
   printf("Numbers in reverse order from %d to 1:\n", num);
   printReverse(&num);
   return 0;
}

void printReverse(int *n)
{
   if (*n <= 0)
      return;
   printf("%d ", *n);
   (*n)--;
   printReverse(n);
}
```

11. Array Sum: Write a recursive function to find the sum of all elements in an array of integers.

Without pointers:

```
#include <stdio.h>

int arraySum(int a[], int size);

int main()
{
   int a[] = {1, 2, 3, 4, 5};
   int size = sizeof(a) / sizeof(a[0]);
   int sum = arraySum(a, size);
   printf("Sum of all elements in the array: %d\n", sum);
   return 0;
}

int arraySum(int a[], int size)
{
   if (size <= 0)
      return 0;
   return a[size - 1] + arraySum(a, size - 1);
}
```

With pointers:

```
#include <stdio.h>

int arraySum(int *a, int size);

int main()
{
   int a[] = {1, 2, 3, 4, 5};
```

```
    int size = sizeof(a) / sizeof(a[0]);
    int sum = arraySum(a, size);
    printf("Sum of all elements in the array: %d\n", sum);
    return 0;
}

int arraySum(int *a, int size)
{
    if (size <= 0)
        return 0;
    return *(a + size - 1) + arraySum(a, size - 1);
}
```

12. Permutations of a String: Develop a recursive function to generate all possible permutations of a given string.

Without pointers:

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

void swap(char str[], int x, int y);
void permutationString(char str[], int l, int r);

int main()
{
    char str[100];
    printf("Enter a string: ");
    scanf("%s", str);
```

```
    int n = strlen(str);
    printf("Permutations of the string are:\n");
    permutationString(str, 0, n - 1);
    return 0;
}

void swap(char str[], int x, int y)
{
    char temp = str[x];
    str[x] = str[y];
    str[y] = temp;
}

void permutationString(char str[], int l, int r)
{
    if (l == r)
        printf("%s\n", str);
    else
    {
        for (int i = l; i <= r; i++)
        {
            swap(str, l, i);
            permutationString(str, l + 1, r);
            swap(str, l, i);
        }
    }
}
```

With pointers:

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

void swap(char *x, char *y);
void permutationString(char *str, int l, int r);

int main()
{
   char str[100];
   printf("Enter a string: ");
   scanf("%s", str);
   int n = strlen(str);
   printf("Permutations of the string are:\n");
   permutationString(str, 0, n - 1);
   return 0;
}

void swap(char *x, char *y)
{
   char t = *x;
   *x = *y;
   *y = t;
}

void permutationString(char *str, int l, int r)
{
   if (l == r)
      printf("%s\n", str);
```

```
        else
        {
            for (int i = l; i <= r; i++)
            {
                swap((str + l), (str + i));
                permutationString(str, l + 1, r);
                swap((str + l), (str + i));
            }
        }
    }
```

Linked list

```
/* 20->14>21->45->89->56->63->72
```

1. Display the linked list
2. Count the number of elements present in the link list and print it
3. Sum of all the elements in the linked list
4. Find the maximum element in the linked list
5. Find the minmum element in the linked list
6. Search for a particullar element whether it is present in the linked list */

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

```
struct Node
{
   int data;
   struct Node *next;
};

void displayNodes(struct Node*);
int countNode(struct Node*);
int sumOfNodes(struct Node *p);
int findMax(struct Node *p);
int findMin(struct Node *p);
void searchNode(struct Node *p);

int main()
{
   struct Node *first = NULL, *t = NULL, *newNode = NULL;
   int a[] = {20, 14, 21, 45, 89, 56, 63, 72};
   int n = sizeof(a) / sizeof(a[0]);
   for (int i = 0; i < n; i++)
   {
      newNode = (struct Node *)malloc(sizeof(struct Node));
```

```
    newNode->data = a[i];
    newNode->next = NULL;
    if (first == NULL)
    {
        first = newNode;
        t = first;
    }
    else
    {
        t->next = newNode;
        t = t->next;
    }
}
printf("Linked list: ");
displayNodes(first);

int c = countNode(first);
printf("Total count = %d\n", c);

int s = sumOfNodes(first);
printf("Sum of all elements = %d\n", s);
```

```
    int max = findMax(first);
    printf("Maximum element = %d\n", max);

    int min = findMin(first);
    printf("Minimum element = %d\n", min);

    searchNode(first);
    return 0;
}

// Function to display the linked list
void displayNodes(struct Node *p)
{
    while(p != NULL)
    {
        printf("%d ", p->data);
        p = p->next;
    }
    printf("\n");
}

// Function to count the number of elements present
```

```
int countNode(struct Node *p)
{
    int count = 0;
    while(p)
    {
        count++;
        p = p->next;
    }
    return count;
}

// Function to find the sum of all elements in the linked list
int sumOfNodes(struct Node *p)
{
    int sum = 0;
    while (p != NULL)
    {
        sum += p->data;
        p = p->next;
    }
    return sum;
}
```

```
// Function to find the maximum element in the linked list
int findMax(struct Node *p) {
    int max = p->data;
    while (p != NULL)
    {
        if (p->data > max)
            max = p->data;
        p = p->next;
    }
    return max;
}

// Function to find the minimum element in the linked list
int findMin(struct Node *p) {
    int min = p->data;
    while (p != NULL)
    {
        if (p->data < min)
            min = p->data;
        p = p->next;
    }
```

```
    return min;
}

// Function to search for a particular element in the linked list
void searchNode(struct Node *p) {
    int i;
    printf("Enter the element to search: ");
    scanf("%d", &i);
    int j = 0, found = 0;
    while (p != NULL)
    {
        if (p->data == i)
        {
            printf("Element %d found at position %d\n", i, j);
            found = 1;
        }
        p = p->next;
        j++;
    }
    if(!found)
        printf("Element %d not found in the linked list\n", i);
}
```