In [ ]:

```
# prompt: Write a program in C to accept dimensions of two matrices and to che
# matrix multiplication is possible, if it is possible then multiply two matri
# using pointers.
# Dont use Dynamic memory allocation

#include <stdio.h>

int main() {
  int rows1, cols1, rows2, cols2;

  printf("Enter the dimensions of the first matrix (rows columns): ");
  scanf("%d %d", &rows1, &cols1);

  printf("Enter the dimensions of the second matrix (rows columns): ");
  scanf("%d %d", &rows2, &cols2);

  if (cols1 != rows2) {
    printf("Matrix multiplication is not possible.\n");
    return 1;
  }

  int matrix1[rows1][cols1], matrix2[rows2][cols2], result[rows1][cols2];

  printf("Enter the elements of the first matrix:\n");
  for (int i = 0; i < rows1; i++) {
    for (int j = 0; j < cols1; j++) {
      scanf("%d", &matrix1[i][j]);
    }
  }

  printf("Enter the elements of the second matrix:\n");
  for (int i = 0; i < rows2; i++) {
    for (int j = 0; j < cols2; j++) {
      scanf("%d", &matrix2[i][j]);
    }
  }

  // Perform matrix multiplication using pointers
  for (int i = 0; i < rows1; i++) {
    for (int j = 0; j < cols2; j++) {
      int sum = 0;
      for (int k = 0; k < cols1; k++) {
        sum += *(*(matrix1 + i) + k) * *(*(matrix2 + k) + j);
      }
      *(*(result + i) + j) = sum;
    }
  }

  printf("Resultant matrix:\n");
  for (int i = 0; i < rows1; i++) {
    for (int j = 0; j < cols2; j++) {
      printf("%d ", result[i][j]);
    }
```

```
    printf("\n");
  }

  return 0;
}
```

In [ ]:
```
# prompt: Write a program in C to print all permutations of a given string usi
# Expected Output :
# The permutations of the string are :
# abcd abdc acbd acdb adcb adbc bacd badc bcad bcda bdca bdac cbad
# cbda cabd cadb cdab cdba db
# ca dbac dcba dcab dacb dabc

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

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

void permute(char *str, int l, int r) {
    if (l == r) {
        printf("%s ", str);
    } else {
        for (int i = l; i <= r; i++) {
            swap((str + l), (str + i));
            permute(str, l + 1, r);
            swap((str + l), (str + i)); // backtrack
        }
    }
}

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

In [ ]:
```
# prompt:  Write a program to implement matrix addition and subtraction using

#include <stdio.h>

int main() {
    int rows, cols;

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

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

    int matrix1[rows][cols], matrix2[rows][cols], sum[rows][cols], diff[rows][

    printf("Enter the elements of the first matrix:\n");
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++) {
            scanf("%d", &matrix1[i][j]);
        }
    }

    printf("Enter the elements of the second matrix:\n");
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++) {
            scanf("%d", &matrix2[i][j]);
        }
    }

    // Matrix addition using pointers
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++) {
            *(*(sum + i) + j) = *(*(matrix1 + i) + j) + *(*(matrix2 + i) + j);
        }
    }

    printf("Sum of matrices:\n");
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++) {
            printf("%d ", sum[i][j]);
        }
        printf("\n");
    }

    // Matrix subtraction using pointers
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++) {
            *(*(diff + i) + j) = *(*(matrix1 + i) + j) - *(*(matrix2 + i) + j)
        }
    }

    printf("Difference of matrices:\n");
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++) {
            printf("%d ", diff[i][j]);
        }
        printf("\n");
    }

    return 0;
}
```

In [ ]:

```c
# prompt: Write a C program to search an element in an array using pointers.
# Input: Enter size of array: 10
```

```c
# Enter elements in array: 10 20 30 40 50 60 70 80 90 100
# Enter element to search: 30
# Output: 30 is found at 3 position.

#include <stdio.h>

int main() {
    int size, i, element, found = 0;
    printf("Enter size of array: ");
    scanf("%d", &size);

    int arr[size];
    printf("Enter elements in array: ");
    for (i = 0; i < size; i++) {
        scanf("%d", &arr[i]);
    }

    printf("Enter element to search: ");
    scanf("%d", &element);

    int *ptr = arr; // Assign the base address of the array to the pointer

    for (i = 0; i < size; i++) {
        if (*(ptr + i) == element) {
            printf("%d is found at %d position.\n", element, i + 1);
            found = 1;
            break; // Exit the loop once the element is found
        }
    }

    if (!found) {
        printf("%d is not found in the array.\n", element);
    }

    return 0;
}
```

In [ ]:

```c
# prompt: Write a program in C to transpose a matrix using pointers.

#include <stdio.h>

int main() {
    int rows, cols, i, j;

    printf("Enter the number of rows: ");
    scanf("%d", &rows);
    printf("Enter the number of columns: ");
    scanf("%d", &cols);

    int matrix[rows][cols], transpose[cols][rows];

    printf("Enter the elements of the matrix:\n");
    for (i = 0; i < rows; i++) {
```

```
        for (j = 0; j < cols; j++) {
            scanf("%d", &matrix[i][j]);
        }
    }

    // Transpose the matrix using pointers
    for (i = 0; i < rows; i++) {
        for (j = 0; j < cols; j++) {
            *(*(transpose + j) + i) = *(*(matrix + i) + j);
        }
    }

    printf("Transposed matrix:\n");
    for (i = 0; i < cols; i++) {
        for (j = 0; j < rows; j++) {
            printf("%d ", transpose[i][j]);
        }
        printf("\n");
    }

    return 0;
}
```

In [ ]:

```
# prompt: A 5-digit positive integer is entered through the keyboard,
# write a function to calculate sum of digits of the 5-digit
# number:
# (1) Without using recursion - call by reference
# (2) Using recursion
# in C

#include <stdio.h>

// Function to calculate the sum of digits without recursion (call by referenc
void sum_digits_no_recursion(int num, int *sum) {
    *sum = 0;
    while (num > 0) {
        *sum += num % 10;
        num /= 10;
    }
}

// Function to calculate the sum of digits using recursion
int sum_digits_recursion(int num) {
    if (num == 0) {
        return 0;
    }
    return (num % 10) + sum_digits_recursion(num / 10);
}

int main() {
    int num;

    printf("Enter a 5-digit positive integer: ");
```

```c
    scanf("%d", &num);

    if (num < 10000 || num > 99999) {
        printf("Invalid input. Please enter a 5-digit number.\n");
        return 1;
    }

    int sum_no_recursion;
    sum_digits_no_recursion(num, &sum_no_recursion);
    printf("Sum of digits (without recursion): %d\n", sum_no_recursion);

    int sum_recursion = sum_digits_recursion(num);
    printf("Sum of digits (using recursion): %d\n", sum_recursion);

    return 0;
}
```

In [ ]:

```c
# prompt: Write a program in C to swap two integers using pointers. Define a f
# that accepts two integer pointers, swaps their values, and displays the resu
# the main function.

#include <stdio.h>

void swap(int *x, int *y) {
    int temp = *x;
    *x = *y;
    *y = temp;
}

int main() {
    int a, b;
    int *ptr_a, *ptr_b;

    printf("Enter two integers: ");
    scanf("%d %d", &a, &b);

    ptr_a = &a;
    ptr_b = &b;

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

    swap(ptr_a, ptr_b);

    printf("After swapping:\n");
    printf("a = %d, b = %d\n", a, b);

    return 0;
}
```

In [ ]:

```c
# prompt: Given three variables x, y, z write a function to circularly shift t
# right. In other words if x = 5, y = 8, z = 10, after circular shift y = 5, z
# =10. Call the function with variables a, b, c to circularly shift values.
```

```python
def circular_shift(x, y, z):
    """Circularly shifts the values of three variables to the right."""
    temp = x
    x = z
    z = y
    y = temp
    return x, y, z

# Example usage:
a = 5
b = 8
c = 10

a, b, c = circular_shift(a, b, c)

print("After circular shift:")
print("a =", a)
print("b =", b)
print("c =", c)
```

In [ ]:

```c
// prompt: Given an array of 5 element write a function in C to circularly shif
// right or left.

#include <stdio.h>

void circular_shift_right(int arr[], int n) {
    int temp = arr[n - 1];
    for (int i = n - 1; i > 0; i--) {
        arr[i] = arr[i - 1];
    }
    arr[0] = temp;
}

void circular_shift_left(int arr[], int n) {
    int temp = arr[0];
    for (int i = 0; i < n - 1; i++) {
        arr[i] = arr[i + 1];
    }
    arr[n - 1] = temp;
}

int main() {
    int arr[5];
    printf("Enter 5 elements of the array: ");
    for (int i = 0; i < 5; i++) {
        scanf("%d", &arr[i]);
    }

    // Circular shift right
    circular_shift_right(arr, 5);
    printf("Circular shift right: ");
    for (int i = 0; i < 5; i++) {
```

```
        printf("%d ", arr[i]);
    }
    printf("\n");

    // Circular shift left
    circular_shift_left(arr, 5);
    printf("Circular shift left: ");
    for (int i = 0; i < 5; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");

    return 0;
}
```

In [ ]:

```
# prompt: Given an array of 5 element write a function in C to circularly shif

#include <stdio.h>

void circular_shift_right(int arr[], int n, int shift) {
    int temp[n];
    for (int i = 0; i < n; i++) {
        temp[i] = arr[i];
    }
    for (int i = 0; i < n; i++) {
        arr[(i + shift) % n] = temp[i];
    }
}

void circular_shift_left(int arr[], int n, int shift) {
    int temp[n];
    for(int i = 0; i < n; i++) {
        temp[i] = arr[i];
    }
    for (int i = 0; i < n; i++) {
        arr[i] = temp[(i + n - (shift % n)) % n];
    }
}


int main() {
    int arr[5];
    int shift;

    printf("Enter 5 elements of the array: ");
    for (int i = 0; i < 5; i++) {
        scanf("%d", &arr[i]);
    }

    printf("Enter the shift value: ");
    scanf("%d", &shift);

    // Circular shift right
```

```
    circular_shift_right(arr, 5, shift);
    printf("Circular shift right: ");
    for (int i = 0; i < 5; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");

    // Circular shift left (reset the array first)
    // Initialize the array again for the left shift
     printf("Enter 5 elements of the array again for left shift: ");
    for (int i = 0; i < 5; i++) {
        scanf("%d", &arr[i]);
    }

    circular_shift_left(arr, 5, shift);
    printf("Circular shift left: ");
    for (int i = 0; i < 5; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");

    return 0;
}
```

In [ ]:

```
# prompt: Write a program in C to find the factorial of a given number using p

#include <stdio.h>

int factorial(int n, int *result) {
    if (n < 0) {
        return 0; // Factorial is not defined for negative numbers
    } else if (n == 0) {
        *result = 1;
        return 1;
    } else {
        *result = 1;
        for (int i = 1; i <= n; i++) {
            *result *= i;
        }
        return 1;
    }
}

int main() {
    int num, fact;
    int *ptr = &fact; // Declare a pointer and assign the address of fact

    printf("Enter a non-negative integer: ");
    scanf("%d", &num);

    if (factorial(num, ptr)) {
        printf("Factorial of %d = %d\n", num, fact);
    } else {
```

```
        printf("Factorial is not defined for negative numbers.\n");
    }
    return 0;
}
```

In [ ]:

```
# prompt: Write a program in C to check Armstrong and Perfect numbers using th
# function.

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

// Function to check if a number is an Armstrong number
int isArmstrong(int num) {
    int originalNum = num;
    int sum = 0;
    int numDigits = 0;
    int temp = num;

    // Count the number of digits
    while (temp != 0) {
        numDigits++;
        temp /= 10;
    }

    temp = num;
    while (temp != 0) {
        int digit = temp % 10;
        sum += pow(digit, numDigits);
        temp /= 10;
    }

    return sum == originalNum;
}

// Function to check if a number is a perfect number
int isPerfect(int num) {
    int sum = 1; // 1 is always a divisor

    for (int i = 2; i <= sqrt(num); i++) {
        if (num % i == 0) {
            sum += i;
            if (i != num / i) {
                sum += num / i;
            }
        }
    }
    return sum == num;
}

int main() {
    int num;
    printf("Enter a positive integer: ");
    scanf("%d", &num);
```

```c
    if (isArmstrong(num)) {
        printf("%d is an Armstrong number.\n", num);
    } else {
        printf("%d is not an Armstrong number.\n", num);
    }

    if (isPerfect(num)) {
        printf("%d is a perfect number.\n", num);
    } else {
        printf("%d is not a perfect number.\n", num);
    }

    return 0;
}
```

In [ ]:

```c
// prompt: Write a C program to demonstrate the use of pointers in function arg
// Your program should:
// 1. Define a function `swap(int *a, int *b)` that swaps the values of two int
// using pointers.
// 2. In the `main()` function, take two integer inputs from the user and call
// `swap()` function.
// 3. Display the values before and after swapping.

#include <stdio.h>

void swap(int *a, int *b) {
    int temp = *a;
    *a = *b;
    *b = temp;
}

int main() {
    int a, b;
    int *ptr_a, *ptr_b;

    printf("Enter two integers: ");
    scanf("%d %d", &a, &b);

    ptr_a = &a;
    ptr_b = &b;

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

    swap(ptr_a, ptr_b);

    printf("After swapping:\n");
    printf("a = %d, b = %d\n", a, b);

    return 0;
}
```

In [ ]:

```c
# prompt: Implement a calculator using function pointers.
# Create separate functions for addition, subtraction, multiplication, and div
# Store function pointers in an array and call functions dynamically based on
# user input. in C

#include <stdio.h>

// Function pointers
typedef int (*operation)(int, int);

// Function definitions
int add(int a, int b) {
  return a + b;
}

int subtract(int a, int b) {
  return a - b;
}

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

int divide(int a, int b) {
  if (b == 0) {
    printf("Error: Division by zero\n");
    return 0; // Or handle the error appropriately
  }
  return a / b;
}

int main() {
  // Array of function pointers
  operation ops[4] = {add, subtract, multiply, divide};

  int choice, num1, num2;

  printf("Simple Calculator using Function Pointers\n");
  printf("1. Add\n");
  printf("2. Subtract\n");
  printf("3. Multiply\n");
  printf("4. Divide\n");
  printf("Enter your choice (1-4): ");
  scanf("%d", &choice);

  if (choice >= 1 && choice <= 4) {
    printf("Enter two numbers: ");
    scanf("%d %d", &num1, &num2);

    // Call the function dynamically
    int result = ops[choice - 1](num1, num2);
    printf("Result: %d\n", result);
  } else {
```

```
    printf("Invalid choice!\n");
  }

  return 0;
}
```

In [ ]:
```
# prompt: Write a program using pointers to compute rhe sum of all interger el
# stored in an array.

#include <stdio.h>

int main() {
    int size;
    printf("Enter the size of the array: ");
    scanf("%d", &size);

    int arr[size];
    printf("Enter the elements of the array:\n");
    for (int i = 0; i < size; i++) {
        scanf("%d", &arr[i]);
    }

    int sum = 0;
    int *ptr = arr; // Pointer to the first element of the array

    for (int i = 0; i < size; i++) {
        sum += *ptr;  // Dereference the pointer to get the value
        ptr++;       // Move the pointer to the next element
    }

    printf("Sum of array elements: %d\n", sum);
    return 0;
}
```

In [ ]:
```
# prompt: Write a program that takes two strings as input and concatenates the
# pointers. Don't use standard library functions for string manipulation. in c

#include <stdio.h>

// Function to concatenate two strings using pointers
void concatenate_strings(char *str1, char *str2) {
    // Find the end of the first string
    while (*str1 != '\0') {
        str1++;
    }

    // Copy the second string to the end of the first string
    while (*str2 != '\0') {
        *str1 = *str2;
        str1++;
        str2++;
    }
```

```
    // Add null terminator to the end of the concatenated string
    *str1 = '\0';
}

int main() {
    char str1[100], str2[100];

    printf("Enter the first string: ");
    scanf("%s", str1);

    printf("Enter the second string: ");
    scanf("%s", str2);

    concatenate_strings(str1, str2);

    printf("Concatenated string: %s\n", str1);

    return 0;
}
```

OPTIONAL QUESTIONS

In [ ]:

```
# prompt: Write a program in C to reverse a string using pointers.

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

void reverse_string(char *str) {
    int len = strlen(str);
    char *start = str;
    char *end = str + len - 1;
    char temp;

    while (start < end) {
        temp = *start;
        *start = *end;
        *end = temp;
        start++;
        end--;
    }
}

int main() {
    char str[100];

    printf("Enter a string: ");
    fgets(str, sizeof(str), stdin); // Use fgets to prevent buffer overflow

    // Remove the trailing newline character from fgets
    str[strcspn(str, "\n")] = 0;

    reverse_string(str);
```

```
    printf("Reversed string: %s\n", str);

    return 0;
}
```

In [ ]:

```
# prompt: Write a function to compute the distance between two points
# and use it to develop another function that will compute the
# area of the triangle whose vertices are A(x1, y1), B(x2, y2),
# and C(x3, y3). Use these functions to develop a function
# which returns a value 1 if the point (x, y) lines inside the
# triangle ABC, otherwise a value 0. in c

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

// Function to compute the distance between two points
double distance(double x1, double y1, double x2, double y2) {
    return sqrt(pow(x2 - x1, 2) + pow(y2 - y1, 2));
}

// Function to compute the area of a triangle
double triangle_area(double x1, double y1, double x2, double y2, double x3, do
    return 0.5 * fabs((x1 * (y2 - y3) + x2 * (y3 - y1) + x3 * (y1 - y2)));
}

// Function to check if a point lies inside a triangle
int is_inside_triangle(double x1, double y1, double x2, double y2, double x3,
    double area_ABC = triangle_area(x1, y1, x2, y2, x3, y3);
    double area_ABP = triangle_area(x1, y1, x2, y2, x, y);
    double area_BCP = triangle_area(x2, y2, x3, y3, x, y);
    double area_CAP = triangle_area(x3, y3, x1, y1, x, y);

    // Check if the sum of the areas of the three smaller triangles equals the
    return (fabs(area_ABC - (area_ABP + area_BCP + area_CAP)) < 1e-6); // Usin
}

int main() {
    double x1, y1, x2, y2, x3, y3, x, y;

    printf("Enter the coordinates of triangle vertices A(x1, y1), B(x2, y2), C
    scanf("%lf %lf %lf %lf %lf %lf", &x1, &y1, &x2, &y2, &x3, &y3);

    printf("Enter the coordinates of the point P(x, y):\n");
    scanf("%lf %lf", &x, &y);

    if (is_inside_triangle(x1, y1, x2, y2, x3, y3, x, y)) {
        printf("1\n");  // Point P lies inside triangle ABC
    } else {
        printf("0\n");  // Point P does not lie inside triangle ABC
    }

    return 0;
}
```

In [ ]:

```c
# prompt: Create a program to reverse a string using pointers.in c

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

void reverse_string(char *str) {
    int len = strlen(str);
    char *start = str;
    char *end = str + len - 1;
    char temp;

    while (start < end) {
        temp = *start;
        *start = *end;
        *end = temp;
        start++;
        end--;
    }
}

int main() {
    char str[100];

    printf("Enter a string: ");
    fgets(str, sizeof(str), stdin); // Use fgets to prevent buffer overflow

    // Remove the trailing newline character from fgets
    str[strcspn(str, "\n")] = 0;

    reverse_string(str);

    printf("Reversed string: %s\n", str);

    return 0;
}
```

In [ ]:

```c
# prompt: write a program to show that pointers can be used to return multiple
# from a function without having to explicitly mention them as return values.i

#include <stdio.h>

// Function to demonstrate returning multiple values using pointers
void calculate_stats(int arr[], int size, int *min, int *max, float *avg) {
    *min = arr[0];
    *max = arr[0];
    int sum = 0;

    for (int i = 0; i < size; i++) {
        if (arr[i] < *min) {
            *min = arr[i];
        }
        if (arr[i] > *max) {
            *max = arr[i];
```

```
        }
        sum += arr[i];
    }

    *avg = (float)sum / size;
}

int main() {
    int arr[] = {5, 2, 9, 1, 5, 6};
    int size = sizeof(arr) / sizeof(arr[0]);
    int min, max;
    float avg;

    // Pass pointers to min, max, and avg to the function
    calculate_stats(arr, size, &min, &max, &avg);

    printf("Minimum: %d\n", min);
    printf("Maximum: %d\n", max);
    printf("Average: %.2f\n", avg);

    return 0;
}
```

In [ ]:

```
# prompt: Write a C program to copy one string to another using pointers.in C

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

void copy_string(char *source, char *destination) {
    while (*source != '\0') {
        *destination = *source;
        source++;
        destination++;
    }
    *destination = '\0'; // Null-terminate the destination string
}

int main() {
    char source_str[100], destination_str[100];

    printf("Enter a string: ");
    scanf("%s", source_str);

    copy_string(source_str, destination_str);

    printf("Copied string: %s\n", destination_str);

    return 0;
}
```

In [ ]:

```
# prompt: Write a C program to modify elements of an array using pointers

#include <stdio.h>
```

```c
int main() {
    int arr[5];
    int *ptr = arr; // Pointer to the array

    printf("Enter 5 elements for the array:\n");
    for (int i = 0; i < 5; i++) {
        scanf("%d", ptr + i); // Accessing elements using pointer arithmetic
    }

    printf("Original array:\n");
    for (int i = 0; i < 5; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");

    printf("Modified array:\n");
    for (int i = 0; i < 5; i++) {
        *(ptr + i) *= 2; // Modify elements using pointer arithmetic
        printf("%d ", *(ptr + i));
    }
    printf("\n");

    return 0;
}
```

In [ ]:

```c
# prompt: Write a function (using pointer parameter) that compares two integer
# see whether they are identical. The function returns 1 if they are identical
# otherwise.in C

#include <stdio.h>

int compare_arrays(int *arr1, int *arr2, int size) {
    for (int i = 0; i < size; i++) {
        if (*(arr1 + i) != *(arr2 + i)) {
            return 0; // Arrays are not identical
        }
    }
    return 1; // Arrays are identical
}

int main() {
    int size;
    printf("Enter the size of the arrays: ");
    scanf("%d", &size);

    int arr1[size], arr2[size];

    printf("Enter elements for the first array:\n");
    for (int i = 0; i < size; i++) {
        scanf("%d", &arr1[i]);
    }
```

```
    printf("Enter elements for the second array:\n");
    for (int i = 0; i < size; i++) {
        scanf("%d", &arr2[i]);
    }

    // Compare the arrays using the function with pointer parameters
    int result = compare_arrays(arr1, arr2, size);

    if (result == 1) {
        printf("The arrays are identical.\n");
    } else {
        printf("The arrays are not identical.\n");
    }

    return 0;
}
```