1. Write a C program to create an integer array of size 5, initialize it with values from 1 to 5, and then use pointer arithmetic to print each element of the array.
   (Topic : Pointer Arthemtic)

```
#include <stdio.h>

int main() {
    int arr[5] = {1, 2, 3, 4, 5};
    int *ptr = arr;

    for(int i = 0; i < 5; i++) {
        printf("%d ", *(ptr + i));
    }
    return 0;
}
```

2. Pointer to Pointer:
   Write a C program to create a pointer to a pointer for an integer variable. Initialize the integer variable with a value, and then print its value using both the single pointer and the pointer to pointer.

```
#include <stdio.h>

int main() {
    int var = 10;
    int *ptr = &var;
    int **ptrToPtr = &ptr;

    printf("Value using single pointer: %d\n", *ptr);
    printf("Value using pointer to pointer: %d\n", **ptrToPtr);

    return 0;
}
```

3. Pointer Function Parameters:
   Write a C function void swap(int *a, int *b) that swaps the values of two integers. Then, write a main function to test this swap function using pointer arguments.

```
#include <stdio.h>

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

int main() {
    int x = 5, y = 10;

    printf("Before swap: x = %d, y = %d\n", x, y);
    swap(&x, &y);
    printf("After swap: x = %d, y = %d\n", x, y);

    return 0;
}
```

4. Dynamic Memory Allocation:
   Write a C program to dynamically allocate memory for an array of integers of size 10. Initialize the array with values from 1 to 10, then print the values and free the allocated memory.

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

int main() {
    int *arr = (int *)malloc(10 * sizeof(int));
    if (arr == NULL) {
        printf("Memory allocation failed\n");
        return 1;
    }

    for(int i = 0; i < 10; i++) {
        arr[i] = i + 1;
    }

    for(int i = 0; i < 10; i++) {
        printf("%d ", arr[i]);
    }

    free(arr);
    return 0;
}
```

5. Pointer to Function:

Write a C program to create a function pointer that points to a function int add(int, int). Use the function pointer to call the add function and print the result.

```
#include <stdio.h>

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

int main() {
    int (*funcPtr)(int, int) = add;
    int result = funcPtr(5, 3);
    printf("Result of addition: %d\n", result);

    return 0;
}
```

6. Functions
    a. Recursive Function:

Write a C function int factorial(int n) that calculates the factorial of a given number using recursion. Test this function in the main program by calculating and printing the factorial of 5.

```
#include <stdio.h>

int factorial(int n) {
    if (n == 0) {
        return 1;
    }
    return n * factorial(n - 1);
}

int main() {
    int num = 5;
    printf("Factorial of %d is %d\n", num, factorial(num));
    return 0;
}
```

7. Function Returning Pointer:

Write a C function int* createArray(int size) that dynamically allocates an array of integers of the given size and returns a pointer to the array. Initialize the array with values from 1 to size and print the array in the main function.

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

int* createArray(int size) {
    int *arr = (int *)malloc(size * sizeof(int));
    if (arr == NULL) {
        printf("Memory allocation failed\n");
        return NULL;
    }

    for(int i = 0; i < size; i++) {
        arr[i] = i + 1;
    }

    return arr;
}

int main() {
    int size = 10;
    int *arr = createArray(size);

    if (arr != NULL) {
        for(int i = 0; i < size; i++) {
            printf("%d ", arr[i]);
        }
        free(arr);
    }

    return 0;
}
```

8. Array of Function Pointers:
   Write a C program to create an array of function pointers, where each function takes two integers as arguments and returns an integer. Include functions for addition, subtraction, multiplication, and division. Use the array to perform these operations on two integers and print the results.

```c
#include <stdio.h>

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) {
        return a / b;
    } else {
        printf("Division by zero error\n");
        return 0;
    }
}

int main() {
    int (*funcArr[])(int, int) = {add, subtract, multiply, divide};
    int x = 10, y = 5;

    printf("Add: %d\n", funcArr[0](x, y));
    printf("Subtract: %d\n", funcArr[1](x, y));
    printf("Multiply: %d\n", funcArr[2](x, y));
    printf("Divide: %d\n", funcArr[3](x, y));

    return 0;
}
```

9. Higher-Order Functions:
   Write a C function void applyFunction(int arr[], int size, void (*func)(int *)) that takes an array, its size, and a pointer to a function that operates on each element of the array. Write a sample function to double the value of each element and use applyFunction to apply it to an array.

```
#include <stdio.h>

void doubleValue(int *n) {
    *n = *n * 2;
}
```

```
void applyFunction(int arr[], int size, void (*func)(int *)) {
    for(int i = 0; i < size; i++) {
        func(&arr[i]);
    }
}

int main() {
    int arr[] = {1, 2, 3, 4, 5};
    int size = sizeof(arr) / sizeof(arr[0]);

    applyFunction(arr, size, doubleValue);

    for(int i = 0; i < size; i++) {
        printf("%d ", arr[i]);
    }

    return 0;
}
```

10. Static Variables in Functions:
    Write a C function that uses a static variable to count how many times the function has been called. Test this function in the main program by calling it multiple times and printing the count.

```
#include <stdio.h>

void countCalls() {
    static int count = 0;
    count++;
    printf("Function called %d times\n", count);
}

int main() {
    countCalls();
    countCalls();
    countCalls();
    return 0;
}
```

11. Structures
    Structure Basics:
    Define a structure struct Point with two integer members x and y. Write a C program to create a Point variable, initialize it with values, and print the values.

```
#include <stdio.h>

struct Point {
    int x;
    int y;
};

int main() {
    struct Point p = {10, 20};
    printf("Point coordinates: x = %d, y = %d\n", p.x, p.y);
    return 0;
}
```

12. Array of Structures:
    Write a C program to define a structure struct Student with members name, age, and marks. Create an array of 3 students, initialize them with values, and print the details of each student.

```
#include <stdio.h>

struct Student {
    char name[50];
    int age;
    float marks;
};

int main() {
    struct Student students[3] = {
        {"John", 20, 85.5},
        {"Alice", 22, 90.0},
        {"Bob", 19, 75.0}
    };

    for(int i = 0; i < 3; i++) {
        printf("Student %d: Name = %s, Age = %d, Marks = %.2f\n", i + 1, students[i].name,
students[i].age, students[i].marks);
    }

    return 0;
}
```

13. Structure Pointers:

Write a C program to define a structure struct Rectangle with members length and width. Create a pointer to a Rectangle variable, dynamically allocate memory for it, initialize the members, and print the values.

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

struct Rectangle {
    int length;
    int width;
};

int main() {
    struct Rectangle *rect = (struct Rectangle *)malloc(sizeof(struct Rectangle));
    if (rect == NULL) {
        printf("Memory allocation failed\n");
        return 1;
    }

    rect->length = 15;
    rect->width = 10;

    printf("Rectangle: length = %d, width = %d\n", rect->length, rect->width);

    free(rect);
    return 0;
}
```

14. Nested Structures:
    Write a C program to define a structure struct Date with members day, month, and year, and another structure struct Student with members name and birthdate of type struct Date. Create a Student variable, initialize it with values, and print the student's details including the birthdate.

```
#include <stdio.h>

struct Date {
    int day;
    int month;
    int year;
};

struct Student {
    char name[50];
    struct Date birthdate;
};

int main() {
    struct Student student = {"John Doe", {15, 6, 1999}};

    printf("Student Name: %s\n", student.name);
    printf("Birthdate: %02d-%02d-%04d\n", student.birthdate.day, student.birthdate.month,
student.birthdate.year);

    return 0;
}
```

15. Structure as Function Argument:
    Write a C function void printStudent(struct Student s) that takes a Student structure as an argument and prints the details. Define a Student structure in the main program, initialize it with values, and call printStudent to print the student's details.

```
#include <stdio.h>

struct Student {
    char name[50];
    int age;
    float marks;
};

void printStudent(struct Student s) {
    printf("Name: %s, Age: %d, Marks: %.2f\n", s.name, s.age, s.marks);
}

int main() {
    struct Student student = {"Alice", 21, 88.5};
    printStudent(student);
    return 0;
```

- }