C Programming Short Notes

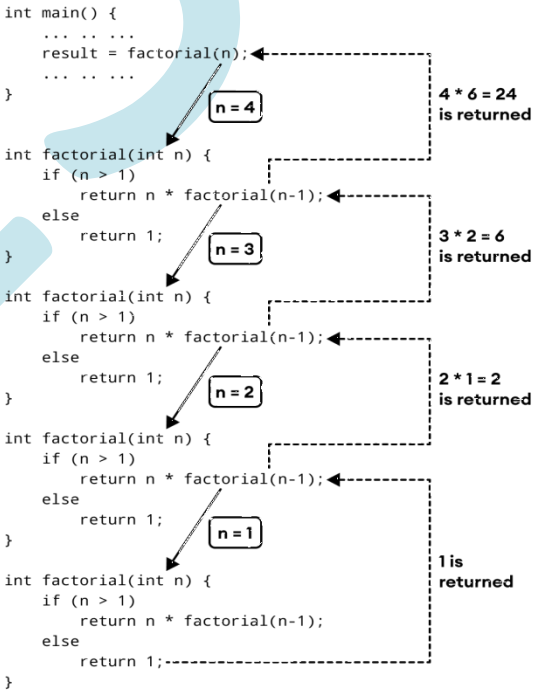
Part 8

1. Recursion

- Recursion is a programming technique where a **function calls itself** directly or indirectly in order to solve a problem.
- It's a powerful tool for solving problems that can be **broken down into smaller, similar subproblems.**
- Recursive solutions often involve **breaking down a problem into simpler versions of the same problem until a base case is reached, at which point the solution can be computed directly.**

To calculate the factorial of a non-negative integer:

```
#include <stdio.h>
int factorial(int n) {
    if (n>1) {
        return n * factorial(n - 1);
    }
    else {
        return 1;
    }
}
int main() {
    int n = 4;
    printf("Factorial of %d is %d\n", n, factorial(n));
    return 0;
}
```



- Recursion can be used to solve a wide range of problems, including **tree and graph traversal, dynamic programming, and divide-and-conquer algorithms.**
- However, it's important to ensure that recursive functions have well-defined base cases to prevent infinite recursion.
- Additionally, excessive recursion can lead to **stack overflow errors**, so **it's important to consider the space complexity of recursive algorithms.**

Types of Recursion

1. **Direct Recursion:**

Direct recursion occurs when a **function calls itself directly.** In other words, the function directly invokes itself within its own body.

Example:

```
void directRecursion(int n) {
   if (n > 0) {
      printf("%d ", n);
      directRecursion(n - 1); // Function calling itself
   }
}
```

2. **Indirect Recursion:**

- Indirect recursion occurs **when two or more functions call each other in a circular manner**, eventually forming a cycle.
- One function calls the second function, which in turn calls the first function or some other function that leads back to the initial caller.

Example:

```
void function1(int n);
void function2(int n) {
   if (n > 0) {
      printf("%d ", n);
      function1(n - 1); // Calling another function
   }
}
void function1(int n) {
   if (n > 0) {
      printf("%d ", n);
      function2(n - 1); // Calling another function
   }
}
```

3. Tail Recursion:

- Tail recursion occurs when **the recursive call is the last operation performed by the function.**
- In other words, **the recursive call is at the end of the function body, and there's no further computation to be done after the recursive call returns.**

Example:

```
void tailRecursion(int n) {
   if (n > 0) {
      printf("%d ", n);
      tailRecursion(n - 1); // Tail recursion
   }
}
```

4. Non-Tail Recursion:

- Non-tail recursion occurs **when there are further computations to be done after the recursive call returns**.
- In this case, the return value of the recursive call is used in additional computations.

Example:

```
int nonTailRecursion(int n) {
   if (n == 0)
      return 0;
   else
      return n + nonTailRecursion(n - 1); // Non-tail recursion
}
```