1. Prime Number Check Function

A prime number is a natural number greater than 1 that has no positive divisors other than 1 and itself[1].

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

int is_prime(int num) {
    if (num <= 1) {
        return 0; // 0 and 1 are not prime
    }
    // Check for divisors up to the square root of num
    for (int i = 2; i <= sqrt(num); i++) {
        if (num % i == 0) {
            return 0; // Not prime (found a divisor) [cite: 47]
        }
    }
    return 1; // Prime [cite: 47]
}

// Example usage in main:
/*
void main() {
    int n = 13;
    if (is_prime(n) == 1)
        printf("%d is Prime.\n", n);
    else
        printf("%d is Not Prime.\n", n);
}
*/
```

---

2. Function to Find Factorial of a Number

Factorial of a non-negative integer N is the product of all positive integers less than or equal to N.

C
```
#include <stdio.h>

long long factorial(int n) {
    long long result = 1;
    if (n < 0) {
        return -1; // Indicate error for negative numbers
    }
```

```
    for (int i = 2; i <= n; i++) {
        result *= i;
    }
    return result; // [cite: 48]
}

// Example usage:
/*
void main() {
    int num = 5;
    printf("Factorial of %d is: %lld\n", num, factorial(num)); // 120
}
*/
```

---

3. Recursive Function and Its Use

- **Definition:** A **recursive function** is a function that calls itself, either directly or indirectly[2].
- **Use:** It is primarily used to solve problems that can be broken down into smaller, self-similar subproblems[3]. This often leads to code that is **cleaner** and **more intuitive** for problems defined recursively, such as **traversals** (tree/graph), **sorting** (Merge Sort/Quick Sort), and **mathematical sequences** (Fibonacci, Factorial)[4].

---

4. Loop That Never Ends (Infinite Loop)

If you create a loop that never ends (an **infinite loop**), the program execution will **hang** or freeze at that point[5].

- The loop will **repeatedly execute** the same block of code indefinitely[6].
- It **consumes CPU time** and system resources, preventing the program from moving to subsequent code or terminating normally.
- The only way to stop it is usually by **forcibly terminating** the program (e.g., Ctrl+C in a terminal, or closing the application window).

---

5 & 9. Storage Class, Scope, Visibility, and Lifetime

A **Storage Class** defines the scope, visibility, lifetime, and initial value of a variable[77].

Storage Class	Keyword	Scope (Visibility)	Lifetime	Default Value	Storage Location
Automatic	auto	Local (within block/function) [8]	Until function/block terminates [9]	Garbage value [10]	Stack [11]
External	extern	Global (across all files) [12]	Entire program execution [13]	Zero (0) [14]	Data Segment [15]
Static	static	Local (if in function); File (if global) [16]	Entire program execution (value persists) [17]	Zero (0) [18]	Data Segment [19]
Register	register	Local (within block/function) [20]	Until function/block terminates [21]	Garbage value [22]	CPU Registers (if available) [23]

Example of static (Value persistence):

C
```
void count_calls() {
    static int count = 0; // Initialized only once
    count++;
    printf("Calls: %d\n", count); // Value persists: 1, then 2, then 3...
}
```

---

6. Program to Check Right-Angled Triangle

A triangle is **right-angled** if the square of the longest side (hypotenuse) is equal to the sum of the squares of the other two sides (Pythagorean theorem: $a^2 + b^2 = c^2$) [24].

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

```
// Function to check if a triangle is right-angled
int is_right_angled(int a, int b, int c) {
    // Square the sides to avoid calling a function multiple times
    long long a2 = (long long)a * a;
    long long b2 = (long long)b * b;
    long long c2 = (long long)c * c;

    // Check all three possible combinations for the hypotenuse
    if (a2 + b2 == c2 || a2 + c2 == b2 || b2 + c2 == a2) {
        return 1; // It is a right-angled triangle
    } else {
        return 0; // Not a right-angled triangle
    }
}

void main() {
    int x = 3, y = 4, z = 5;
    if (is_right_angled(x, y, z)) {
        printf("The triangle is right-angled.\n");
    } else {
        printf("The triangle is not right-angled.\n");
    }
}
```

---

7. Program to Print Pattern

The pattern involves printing the character corresponding to the row number, repeated by the row number (A, BB, CCC, etc.)[25].

C

```
#include <stdio.h>

void main() {
    int i, j;
    int rows = 5;
    char start_char = 'A';

    for (i = 0; i < rows; i++) {
        // char_to_print: 'A' + 0 = 'A', 'A' + 1 = 'B', etc.
        char char_to_print = start_char + i;

        for (j = 0; j <= i; j++) {
            printf("%c", char_to_print);
        }
        printf("\n");
    }
```

}

Output:

A
BB
CCC
DDDD
EEEEE

---

8. Function Definition, Advantages, and Swap Program

Define Function

A **function** is a self-contained block of code designed to perform a specific, well-defined task[26].

Advantages of Using Functions

1. **Reusability:** Code written once can be called many times[27].
2. **Modularity:** Breaking a large program into smaller, manageable functions makes the code easier to understand and maintain[28].
3. **Easier Debugging:** Isolating problems to specific functions simplifies the debugging process.

Program to Interchange Contents (Swap) using Function (Call by Reference)

C

```
#include <stdio.h>

// Function to swap the contents using pointers (call by reference)
void swap(int *a, int *b) {
   int temp;
   temp = *a; // Store content of the address pointed to by a
   *a = *b;   // Assign content of b to a
   *b = temp; // Assign original content of a to b
}

void main() {
   int x = 10, y = 20;
   printf("Before swap: x = %d, y = %d\n", x, y);

   // Pass the addresses (&x, &y) of the variables
   swap(&x, &y);
```

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

---

10. Recursion Concept and Factorial

- **Concept of Recursion:** Recursion involves a function calling itself. This continues until a specific termination condition is met[29].
- **Base Condition:** The **base condition** (or termination condition) is a non-recursive path within the function that stops the recursion from continuing infinitely. It specifies the simplest case for which the answer is known directly[30]. Without a base condition, the function would result in an infinite loop and a **stack overflow error**.

Recursive Factorial Function

$$N! = N \times (N-1)!$$
Base Condition: $1! = 1$ and $0! = 1$.

C
```
long long recursive_factorial(int N) {
    if (N == 0 || N == 1) {
        return 1; // Base condition
    } else {
        // Recursive step
        return N * recursive_factorial(N - 1); [cite: 61]
    }
}
```

---

11. Distinctions

(a) Actual and Formal Arguments

Feature	Actual Arguments	Formal Arguments
Definition	Values/variables passed **during the function call**[31].	Variables declared **in the function definition** to receive the actual values[32].
Location	Calling function (e.g., main())	Function definition or prototype

Usage	swap(&x, &y) where x and y are actual arguments	void swap(int *a, int *b) where a and b are formal arguments

(b) Global and Local Variables

Feature	Global Variables	Local Variables
Declaration	Outside all functions [33]	Inside a function or block [34]
Scope	Accessible throughout the entire program (file scope or external scope) [35]	Accessible only within the function/block where they are declared [36]
Lifetime	Entire program execution	Until the function/block finishes execution

(c) Automatic and Static Variables

Feature	Automatic Variables	Static Variables
Keyword	auto (default for locals) [37]	static [38]
Lifetime	Created on function entry, destroyed on exit [39]	Created at program start, exists until program termination (value persists) [40]
Initialization	Garbage value by default [41]	Initialized to zero (0) by default [42]

---

12. The main() Function vs. Other User-Defined Functions

The main() function is the **starting point** of program execution, which is not true for any other user-defined function[43].

- **Starting Point:** The operating system/compiler always looks for and executes main() first.
- **Call Flow:** Other user-defined functions must be **explicitly called** (directly or indirectly) from main() or another function to execute their code[44].
- **Return Value:** main() typically returns an integer (int) to the operating system, indicating the program's exit status (0 usually means successful execution).

---

13. Prototyping

- **Definition: Function Prototyping** (or Function Declaration) is providing the compiler with the function's signature **before** its actual definition[45].
- **Signature includes:** Function name, return type, and the number and type of its parameters[46].
- **Necessity:** It tells the compiler how a function will be called, allowing it to perform **type checking** and ensure that the arguments passed in the function call match the expected types and count. This prevents errors when the function is defined later in the file or in a separate file[47].

Syntax: return_type function_name(data_type param1, data_type param2);

---

14. Function to Return Month Name

C

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

// Function that takes month number (m) and returns the name
const char* get_month_name(int m) {
    // Array of string literals (read-only)
    static const char *months[] = {
        "Invalid Month", "January", "February", "March",
        "April", "May", "June", "July", "August",
        "September", "October", "November", "December"
    };

    if (m >= 1 && m <= 12) {
        return months[m]; // Returns the corresponding name [cite: 68, 69]
    } else {
        return months[0];
    }
```

```
}

void main() {
    int month_num = 3;
    printf("Month number %d is: %s\n", month_num, get_month_name(month_num)); //
Output: March [cite: 69]

    month_num = 1;
    printf("Month number %d is: %s\n", month_num, get_month_name(month_num)); //
Output: January
}
```