7. Recursion

7.1 Understanding Recursion

Recursion is a programming technique where a function calls itself. Every recursive function needs:

1. **Base case:** Condition that stops the recursion
2. **Recursive case:** Function calls itself with modified parameters

7.2 Python vs C Recursion

Python Factorial:

```
def factorial(n):
    if n == 0 or n == 1:
        return 1
    else:
        return n * factorial(n - 1)
```

C Factorial:

```
#include <stdio.h>

int factorial(int n) {
    // Base case
    if (n == 0 || n == 1) {
        return 1;
    }
    // Recursive case
    else {
        return n * factorial(n - 1);
    }
}

int main() {
    int num = 5;
    printf("%d! = %d\n", num, factorial(num));  // Output: 5! = 120
```

```
    return 0;
}
```

7.3 How Recursion Works

Factorial(5) execution trace:

```
factorial(5) = 5 * factorial(4)
             = 5 * 4 * factorial(3)
             = 5 * 4 * 3 * factorial(2)
             = 5 * 4 * 3 * 2 * factorial(1)
             = 5 * 4 * 3 * 2 * 1
             = 120
```

7.4 More Recursive Examples

Fibonacci Sequence

```
#include <stdio.h>

int fibonacci(int n) {
    // Base cases
    if (n == 0) return 0;
    if (n == 1) return 1;

    // Recursive case
    return fibonacci(n - 1) + fibonacci(n - 2);
}

int main() {
    printf("Fibonacci sequence:\n");
    for (int i = 0; i < 10; i++) {
        printf("F(%d) = %d\n", i, fibonacci(i));
    }
    return 0;
}
```

Power Calculation

```c
#include <stdio.h>

double power(double base, int exponent) {
    // Base case
    if (exponent == 0) {
        return 1.0;
    }

    // Handle negative exponents
    if (exponent < 0) {
        return 1.0 / power(base, -exponent);
    }

    // Recursive case
    return base * power(base, exponent - 1);
}

int main() {
    printf("2^5 = %.2f\n", power(2.0, 5));    // Output: 32.00
    printf("3^-2 = %.4f\n", power(3.0, -2));  // Output: 0.1111
    return 0;
}
```

Sum of Array Elements (Preview for next module)

```c
#include <stdio.h>

int sum_array(int arr[], int size) {
    // Base case
    if (size == 0) {
        return 0;
    }

    // Recursive case
    return arr[size - 1] + sum_array(arr, size - 1);
}

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

```
    printf("Sum = %d\n", sum_array(numbers, array_size));  // Output: 15
    return 0;
}
```

7.5 Recursion vs Iteration

Advantages of Recursion:

- More natural for certain problems (tree traversal, mathematical definitions)
- Cleaner, more readable code for some algorithms
- Elegant solution for divide-and-conquer problems

Disadvantages of Recursion:

- Higher memory usage (function call stack)
- Slower execution due to function call overhead
- Risk of stack overflow for deep recursion
- It can causes Crash or freezes when things go wrong (the reasons are already mentioned right above)

When to Use Recursion:

- Mathematical sequences (factorial, fibonacci)
- Tree and graph algorithms
- Divide and conquer problems
- When the problem naturally breaks down into similar subproblems

---

Revision #2
Created 2025-09-06 10:25:58 UTC by DS
Updated 2025-09-06 10:27:37 UTC by DS