# C Programming - Deck 20

## Recursion

Prof. Jyotiprakash Mishra
mail@jyotiprakash.org

# What is Recursion?

- A function that calls itself
- Solves problem by breaking it into smaller subproblems
- Must have two components:
  - **Base case**: Condition to stop recursion
  - **Recursive case**: Function calls itself with smaller input
- Without base case: infinite recursion (stack overflow)
- Each call creates new stack frame
- Stack unwinds as calls return
- Elegant solution for certain problems

# Recursion vs Iteration

| Aspect | Recursion | Iteration |
|---|---|---|
| Approach | Function calls itself | Loop repeats |
| Termination | Base case | Loop condition |
| Stack usage | High (each call) | Low (single frame) |
| Performance | Slower (overhead) | Faster |
| Code | Often cleaner | Can be complex |
| Memory | More (stack frames) | Less |
| Best for | Tree/graph problems | Simple repetition |
| Risk | Stack overflow | Infinite loop |

# Program 1: Simple Countdown Recursion

```c
#include <stdio.h>
void countdown(int n) {
  if (n == 0) {
    printf("Done!\n");
    return;
  }
  printf("%d\n", n);
  countdown(n - 1);
}
int main() {
  countdown(5);
  return 0;
}
```

## Output:

```
5
4
3
2
1
Done!
```

Base case: n == 0, Recursive: countdown(n-1)

# Program 2: Factorial Recursion

```c
#include <stdio.h>
int factorial(int n) {
  if (n == 0 || n == 1) {
    return 1;
  }
  return n * factorial(n - 1);
}
int main() {
  int n = 5;
  printf("Factorial of %d = %d\n",
    n, factorial(n));
  printf("5! = 5*4*3*2*1 = %d\n",
    factorial(5));
  return 0;
}
```

## Output:

```
Factorial of 5 = 120
5! = 5*4*3*2*1 = 120
```

n! = n * (n-1)!

# Program 3: Fibonacci Recursion

```c
#include <stdio.h>
int fibonacci(int n) {
  if (n == 0) return 0;
  if (n == 1) return 1;
  return fibonacci(n - 1) + fibonacci(n - 2);
}
int main() {
  int i;
  printf("Fibonacci series (0-10):\n");
  for (i = 0; i <= 10; i++) {
    printf("%d ", fibonacci(i));
  }
  printf("\n");
  return 0;
}
```

**Output:**

```
Fibonacci series (0-10):
0 1 1 2 3 5 8 13 21 34 55
```

fib(n) = fib(n-1) + fib(n-2)

# Program 4: Sum of Natural Numbers

```c
#include <stdio.h>
int sum(int n) {
  if (n == 0) {
    return 0;
  }
  return n + sum(n - 1);
}
int main() {
  int n = 10;
  printf("Sum of 1 to %d = %d\n", n, sum(n));
  printf("Sum of 1 to 5 = %d\n", sum(5));
  return 0;
}
```

**Output:**

```
Sum of 1 to 10 = 55
Sum of 1 to 5 = 15
```

sum(n) = n + sum(n-1)

# Program 5: Power Function Recursion

```c
#include <stdio.h>
int power(int base, int exp) {
  if (exp == 0) {
    return 1;
  }
  return base * power(base, exp - 1);
}
int main() {
  printf("2^5 = %d\n", power(2, 5));
  printf("3^4 = %d\n", power(3, 4));
  printf("5^3 = %d\n", power(5, 3));
  return 0;
}
```

## Output:

```
2^5 = 32
3^4 = 81
5^3 = 125
```

$\text{base}^{e}xp = base * base^{(}exp - 1)$

# Program 6: GCD Using Euclidean Algorithm

```c
#include <stdio.h>
int gcd(int a, int b) {
  if (b == 0) {
    return a;
  }
  return gcd(b, a % b);
}
int main() {
  printf("GCD(48, 18) = %d\n", gcd(48, 18));
  printf("GCD(100, 25) = %d\n", gcd(100, 25));
  printf("GCD(35, 14) = %d\n", gcd(35, 14));
  return 0;
}
```

**Output:**

```
GCD(48, 18) = 6
GCD(100, 25) = 25
GCD(35, 14) = 7
```

gcd(a,b) = gcd(b, a mod b)

# Program 7: Print Array Using Recursion

```c
#include <stdio.h>
void printArray(int arr[], int n) {
  if (n == 0) {
    return;
  }
  printArray(arr, n - 1);
  printf("%d ", arr[n - 1]);
}
int main() {
  int arr[] = {10, 20, 30, 40, 50};
  printf("Array: ");
  printArray(arr, 5);
  printf("\n");
  return 0;
}
```

## Output:

```
Array: 10 20 30 40 50
```

Recursively traverse array

# Program 8: Sum of Array Elements

```c
#include <stdio.h>
int arraySum(int arr[], int n) {
  if (n == 0) {
    return 0;
  }
  return arr[n - 1] + arraySum(arr, n - 1);
}
int main() {
  int arr[] = {5, 10, 15, 20, 25};
  printf("Sum = %d\n", arraySum(arr, 5));
  int arr2[] = {1, 2, 3, 4, 5};
  printf("Sum = %d\n", arraySum(arr2, 5));
  return 0;
}
```

$\text{sum(arr,n)} = \text{arr[n-1]} + \text{sum(arr,n-1)}$

**Output:**

```
Sum = 75
Sum = 15
```

# Program 9: Find Maximum in Array

```c
#include <stdio.h>
int findMax(int arr[], int n) {
  if (n == 1) {
    return arr[0];
  }
  int max = findMax(arr, n - 1);
  if (arr[n - 1] > max) {
    return arr[n - 1];
  }
  return max;
}
int main() {
  int arr[] = {12, 45, 23, 67, 34};
  printf("Maximum = %d\n", findMax(arr, 5));
  return 0;
}
```

## Output:

```
Maximum = 67
```

Compare current with max of rest

# Program 10: Reverse Array Using Recursion

```c
#include <stdio.h>
void reverse(int arr[], int start, int end) {
  if (start >= end) {
    return;
  }
  int temp = arr[start];
  arr[start] = arr[end];
  arr[end] = temp;
  reverse(arr, start + 1, end - 1);
}
int main() {
  int arr[] = {1, 2, 3, 4, 5};
  int i;
  reverse(arr, 0, 4);
  printf("Reversed: ");
  for (i = 0; i < 5; i++) {
    printf("%d ", arr[i]);
  }
  printf("\n");
  return 0;
}
```

**Output:**

```
Reversed: 5 4 3 2 1
```

Swap ends and recurse inward

# Program 11: Binary Search Recursion

```
#include <stdio.h>
int binarySearch(int arr[], int l, int r, int x) {
  if (l > r) return -1;
  int mid = l + (r - l) / 2;
  if (arr[mid] == x) return mid;
  if (arr[mid] > x)
    return binarySearch(arr, l, mid - 1, x);
  return binarySearch(arr, mid + 1, r, x);
}
int main() {
  int arr[] = {10, 20, 30, 40, 50};
  int result = binarySearch(arr, 0, 4, 30);
  if (result != -1)
    printf("Found at index %d\n", result);
  else
    printf("Not found\n");
  return 0;
}
```

**Output:**

```
Found at index 2
```

Divide and conquer search

# Program 12: String Length Recursion

```c
#include <stdio.h>
int stringLength(char *str) {
  if (*str == '\0') {
    return 0;
  }
  return 1 + stringLength(str + 1);
}
int main() {
  char str1[] = "Hello";
  char str2[] = "Recursion";
  printf("Length of '%s' = %d\n",
    str1, stringLength(str1));
  printf("Length of '%s' = %d\n",
    str2, stringLength(str2));
  return 0;
}
```

## Output:

```
Length of 'Hello' = 5
Length of 'Recursion' = 9
```

Count chars until null terminator

# Program 13: String Reversal Recursion

```c
#include <stdio.h>
void reverseString(char *str, int start, int end) {
  if (start >= end) {
    return;
  }
  char temp = str[start];
  str[start] = str[end];
  str[end] = temp;
  reverseString(str, start + 1, end - 1);
}
int main() {
  char str[] = "Hello";
  printf("Original: %s\n", str);
  reverseString(str, 0, 4);
  printf("Reversed: %s\n", str);
  return 0;
}
```

**Output:**

```
Original: Hello
Reversed: olleH
```

Swap from both ends

```c
#include <stdio.h>
int isPalindrome(char *str, int start, int end) {
  if (start >= end) {
    return 1;
  }
  if (str[start] != str[end]) {
    return 0;
  }
  return isPalindrome(str, start + 1, end - 1);
}
int main() {
  char str1[] = "madam";
  char str2[] = "hello";
  printf("'%s' is %spalindrome\n", str1,
    isPalindrome(str1, 0, 4) ? "" : "not ");
  printf("'%s' is %spalindrome\n", str2,
    isPalindrome(str2, 0, 4) ? "" : "not ");
  return 0;
}
```

**Output:**

```
'madam' is palindrome
'hello' is not palindrome
```

Compare from both ends

# Program 15: Tower of Hanoi

```c
#include <stdio.h>
void hanoi(int n, char from, char to,
  char aux) {
  if (n == 1) {
    printf("Move disk 1 from %c to %c\n",
      from, to);
    return;
  }
  hanoi(n - 1, from, aux, to);
  printf("Move disk %d from %c to %c\n",
    n, from, to);
  hanoi(n - 1, aux, to, from);
}
int main() {
  int n = 3;
  printf("Tower of Hanoi (%d disks):\n", n);
  hanoi(n, 'A', 'C', 'B');
  return 0;
}
```

## Output:

```
Tower of Hanoi (3 disks):
Move disk 1 from A to C
Move disk 2 from A to B
Move disk 1 from C to B
Move disk 3 from A to C
Move disk 1 from B to A
Move disk 2 from B to C
Move disk 1 from A to C
```

Classic recursive problem

# Program 16: Count Digits Recursion

```c
#include <stdio.h>
int countDigits(int n) {
  if (n == 0) {
    return 0;
  }
  return 1 + countDigits(n / 10);
}
int main() {
  printf("Digits in 12345: %d\n",
    countDigits(12345));
  printf("Digits in 999: %d\n",
    countDigits(999));
  printf("Digits in 7: %d\n",
    countDigits(7));
  return 0;
}
```

**Output:**

```
Digits in 12345: 5
Digits in 999: 3
Digits in 7: 1
```

Divide by 10 recursively

# Program 17: Sum of Digits Recursion

```c
#include <stdio.h>
int sumDigits(int n) {
  if (n == 0) {
    return 0;
  }
  return (n % 10) + sumDigits(n / 10);
}
int main() {
  printf("Sum of digits in 123: %d\n",
    sumDigits(123));
  printf("Sum of digits in 999: %d\n",
    sumDigits(999));
  printf("Sum of digits in 4567: %d\n",
    sumDigits(4567));
  return 0;
}
```

## Output:

```
Sum of digits in 123: 6
Sum of digits in 999: 27
Sum of digits in 4567: 22
```

Add last digit + sum of rest

# Program 18: Decimal to Binary Recursion

```c
#include <stdio.h>
void decimalToBinary(int n) {
  if (n == 0) {
    return;
  }
  decimalToBinary(n / 2);
  printf("%d", n % 2);
}
int main() {
  printf("Binary of 10: ");
  decimalToBinary(10);
  printf("\nBinary of 25: ");
  decimalToBinary(25);
  printf("\nBinary of 7: ");
  decimalToBinary(7);
  printf("\n");
  return 0;
}
```

## Output:

```
Binary of 10: 1010
Binary of 25: 11001
Binary of 7: 111
```

Print bits in reverse order

# Program 19: Print N to 1 and 1 to N

```c
#include <stdio.h>
void printDescending(int n) {
  if (n == 0) return;
  printf("%d ", n);
  printDescending(n - 1);
}
void printAscending(int n) {
  if (n == 0) return;
  printAscending(n - 1);
  printf("%d ", n);
}
int main() {
  printf("Descending: ");
  printDescending(5);
  printf("\nAscending: ");
  printAscending(5);
  printf("\n");
  return 0;
}
```

## Output:

```
Descending: 5 4 3 2 1
Ascending: 1 2 3 4 5
```

Print before vs after recursive call

# Program 20: Recursion vs Iteration Comparison

```c
#include <stdio.h>
int factorialRecursive(int n) {
  if (n <= 1) return 1;
  return n * factorialRecursive(n - 1);
}
int factorialIterative(int n) {
  int result = 1;
  int i;
  for (i = 2; i <= n; i++) {
    result *= i;
  }
  return result;
}
int main() {
  int n = 5;
  printf("Recursive: %d! = %d\n",
    n, factorialRecursive(n));
  printf("Iterative: %d! = %d\n",
    n, factorialIterative(n));
  printf("Both give same result\n");
  return 0;
}
```

## Output:

```
Recursive: 5! = 120
Iterative: 5! = 120
Both give same result
```

Same result, different approaches

# Advantages of Recursion

- Clean and elegant code for certain problems
- Natural for tree and graph traversal
- Simplifies divide-and-conquer algorithms
- Easier to understand for some problems
- Reduces complex problems to simpler ones
- Perfect for problems with recursive structure
- Examples: Tree traversal, backtracking, Tower of Hanoi

# Disadvantages of Recursion

- Higher memory usage (stack frames)
- Slower than iteration (function call overhead)
- Risk of stack overflow with deep recursion
- Can be harder to debug
- May recalculate same values (like Fibonacci)
- Not always the most efficient solution
- Stack size is limited by system

# Key Takeaways

- Recursion: function calls itself
- Must have base case (termination condition)
- Recursive case reduces problem size
- Each call creates new stack frame
- Stack unwinds as functions return
- Without base case: infinite recursion
- Good for: trees, divide-and-conquer, backtracking
- Trade-off: elegance vs performance
- Always consider iterative alternative
- Understanding recursion is fundamental to CS