Computer Programming using C

Recursion

Instructor: HOU, Fen

2025

Recursion(递归）

recursion

- Read the following program

```
#include <stdio.h>
void print(int n);
int main()
{
print(5);
return 0;
}
void print(int n)
 {
printf("%d ", n);

if(n <= 1)
{
return; /* Return and make no more recursive call */
}
print(n - 1);
}
```

Output:
5 4 3 2 1

What is Recursion?

- A function can call itself and such a process is called recursion

- Solve a problem by solving a smaller version of the exact same problem first.

- Recursion is a way that solves a problem by solving a smaller problem of the same type.

Problem Solved by Closed Form

- Example, n factorial (n!)
  - Closed form solution

$$n! = \begin{cases} 1 & \text{if } n = 0 \\ 1*2*3*\ldots*(n-1)*n & \text{if } n > 0 \end{cases}$$

Code with the Closed Form

- Code to solve n factorial

```
#include <stdio.h>
int fact(int n);
int main(void)
{
  int i=5;
  printf("count is %d", fact(i));
  return 0;
}

int fact(int n)
{
  int t, answer;
   answer = 1;
  for(t=1; t<=n; t++)
    answer=answer*t;
  return answer;
}
```

Solved by the closed form

Problems Solved Recursively

- There are many problems whose solution can be defined recursively.
- Example, n factorial (n!)
  - Recursive solution

$$n! = \begin{cases} 1 & \text{if } n = 0 \\ (n-1)! * n & \text{if } n > 0 \end{cases}$$

Code with Recursion

- Code to solve n factorial by recursion

```
#include <stdio.h>
int fact(int n);
int main(void)
{
 int i=5;
 printf("count is %d", fact(i));
 return 0;
}
```

```
int fact(int n)
{
 int answer;
  if(n==0) return 1;
 /* recursive call */
 answer = fact(n-l)*n;
 return answer ;
}
```

Solved by recursion

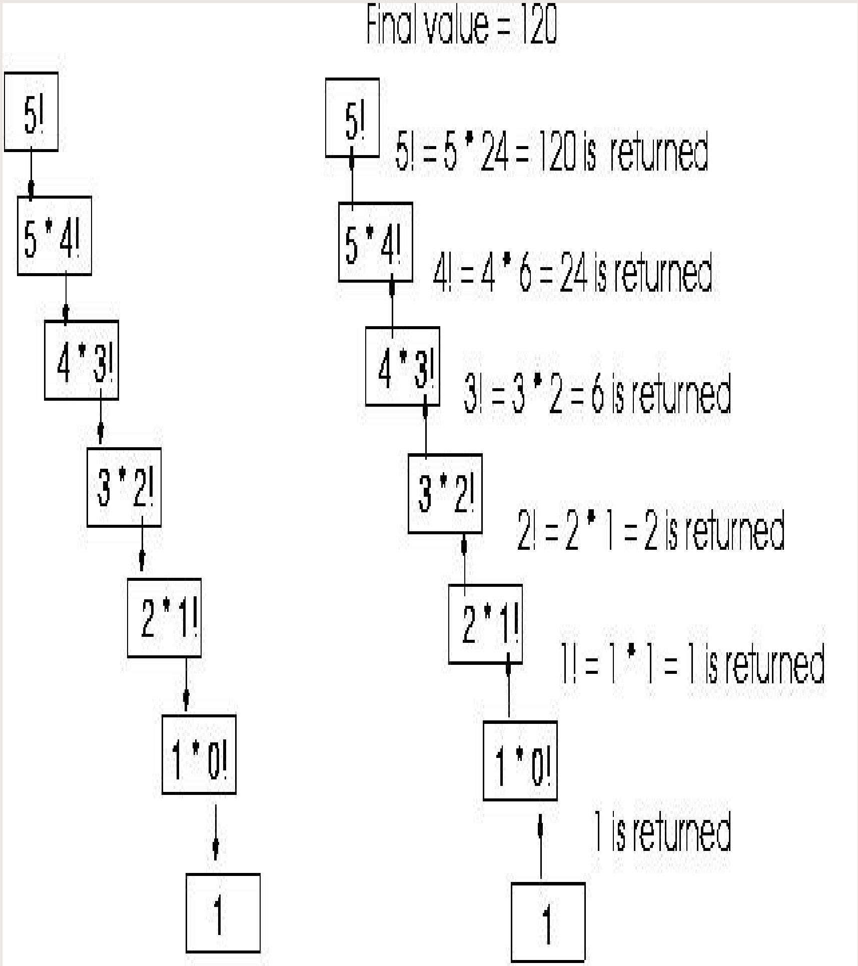
Coding the Factorial Function

- Recursive function code:

```
int fact(int n)
{
  int answer;
    if(n==0) return 1;  /*base case*/
  /* recursive call */
  answer = fact(n-l)*n;
  return answer ;
}
```

Factorial Execution

```
int fact(int n)
{
 int answer;
   if(n==0) return 1;  /*base case*/
 /* recursive call */
 answer = fact(n-l)*n;
 return answer ;
}
```



Factorial Execution

- First, the recursive function calls will be proceed:

  fact(5)=5*fact(4)

  fact(4)=4*fact(3)

  fact(3)=3*fact(2)

  fact(2)=2*fact(1)

  fact(1)=1*fact(0)

- The actual values return in the reverse order

  fact(0)=1

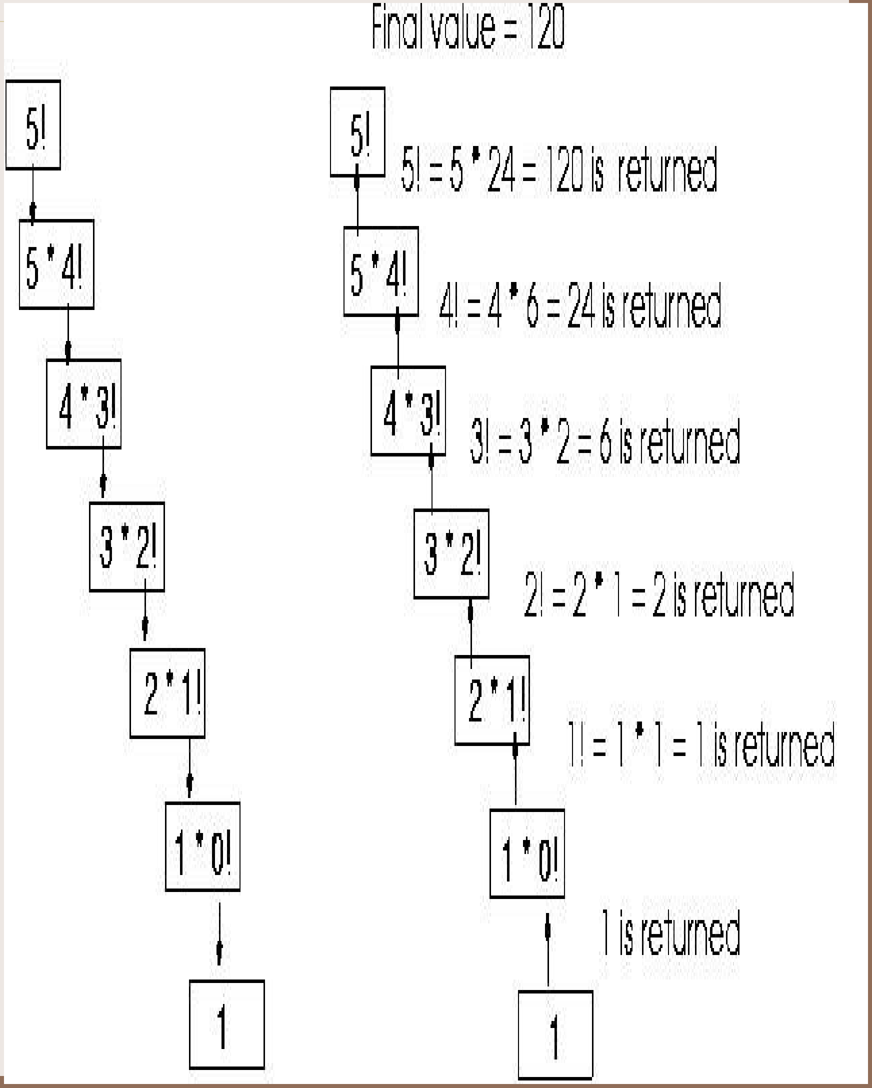
  fact(1)=1*1=1

  fact(2)=2*1=2

  fact(3)=3*2=6

  fact(4)=4*6=24

  fact(5)=5*24=120



Write a Recursive Function

- Determine the size factor;
- Determine the base case(s): that is the one for which you know the answer.
- Determine the general case(s):that is the one where the answer is expressed as a smaller version of itself.

Example: Closed-form Solution

- n choose k (Combinations): Given n items, how many different sets with size k items can be chosen?
- Closed-form solution:

$$\binom{n}{k} = \frac{n!}{k!(n-k)!} \quad , \ 1 < k < n$$

with base cases

$$\binom{n}{1} = n \ \ (k = 1), \binom{n}{n} = 1 \ \ (k = n)$$

Example: n choose k (Combination)

- n choose k (Combinations): Given n items, how many different sets with size k items can be chosen?
- Recursive solution:

$$\binom{n}{k} = \binom{n-1}{k} + \binom{n-1}{k-1} \quad , \quad 1 < k < n$$

with base cases

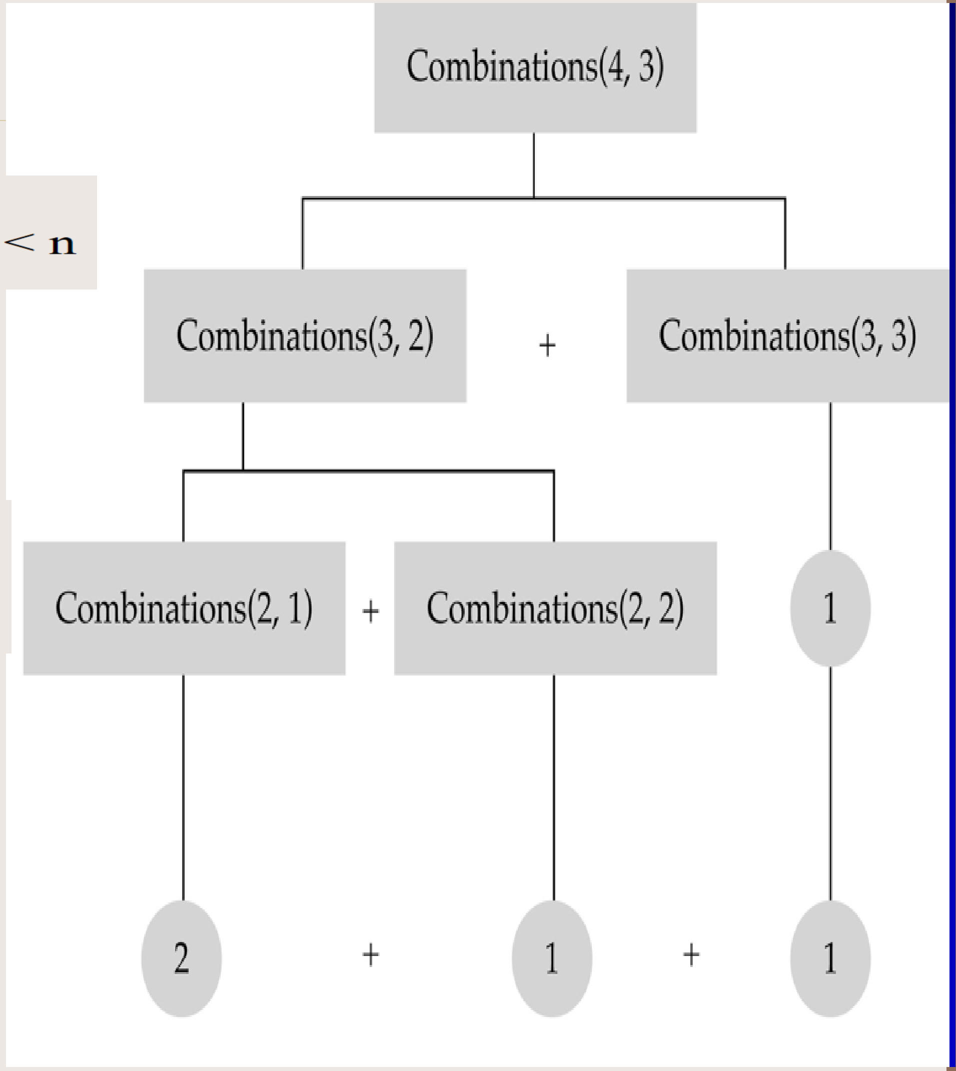
$$\binom{n}{1} = n \ \ (k = 1), \binom{n}{n} = 1 \ \ (k = n)$$

Example: n=4,k=3

$$\binom{n}{k} = \binom{n-1}{k} + \binom{n-1}{k-1}, \quad 1 < k < n$$

with base cases

$$\binom{n}{1} = n \quad (k = 1), \quad \binom{n}{n} = 1 \quad (k = n)$$



n Choose k (Combination)

Exercise: Write the recursive function code for the combination.

```
int Combinations(int n, int k)
  {
  .......
   }
```

$$\binom{n}{k} = \binom{n-1}{k} + \binom{n-1}{k-1} \quad , \quad 1 < k < n$$

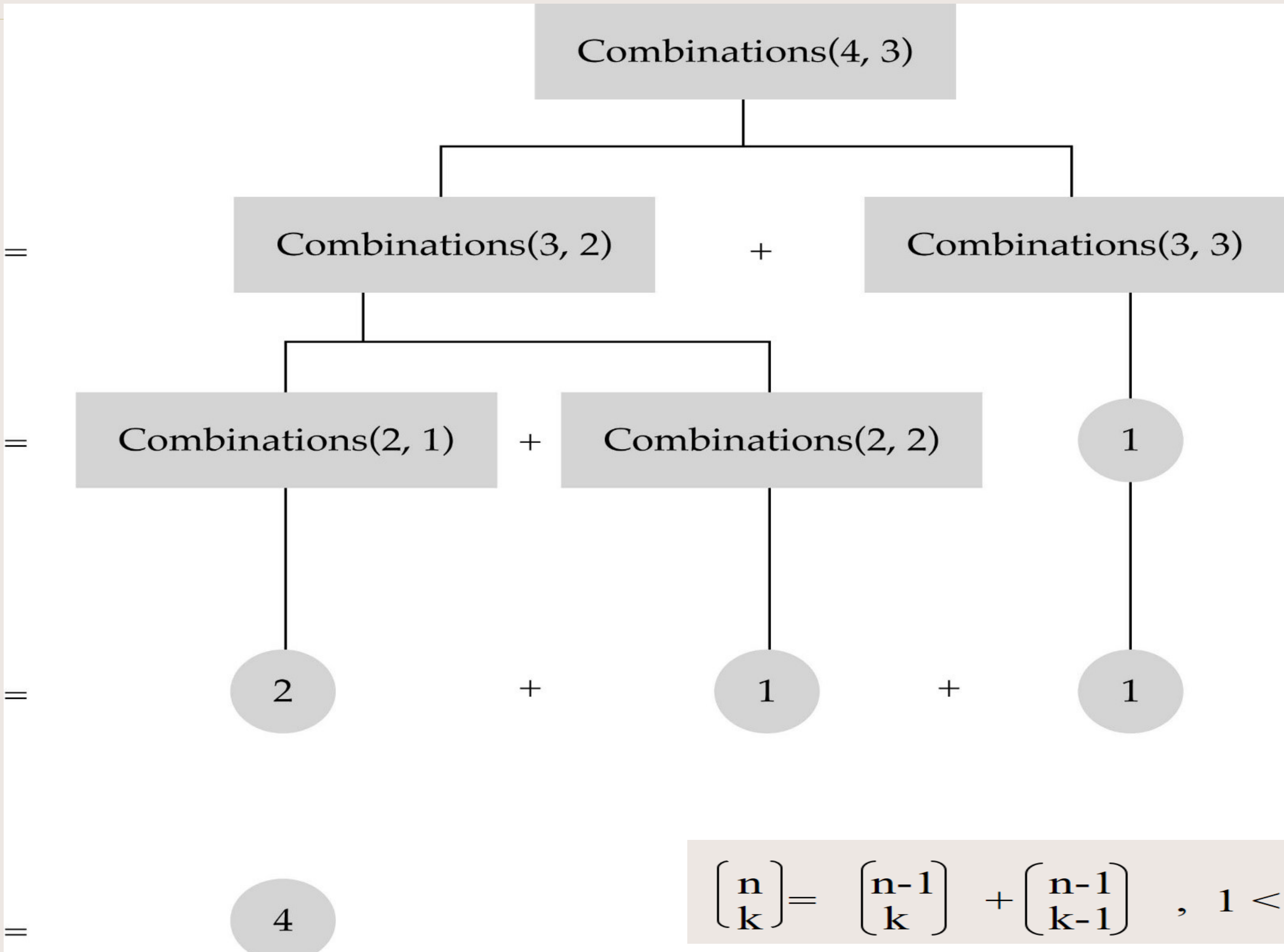
n Choose k (Combination)

- Recursive Function Code

```
int Combinations(int n, int k)
{
int answer;
if(k == 1)  /*base case 1*/
   return n;
 else if (n == k)  /* base case 2*/
             return 1;
          else
      answer = Combinations(n-1, k) + Combinations(n-1, k-1);
 return answer;
}
```

$$\binom{n}{k} = \binom{n-1}{k} + \binom{n-1}{k-1} \quad , \quad 1 < k < n$$

n Choose k (Combination)



$$\begin{pmatrix} n \\ k \end{pmatrix} = \begin{pmatrix} n-1 \\ k \end{pmatrix} + \begin{pmatrix} n-1 \\ k-1 \end{pmatrix} , \quad 1 < k < n$$

Verify a Recursive Function

Use the “Three-Question-Method” to verify the recursive function.

1. The Base-case question:

   Is there a nonrecursive way out of the function, and does the routine work correctly for this "base" case?

2. The Smaller-caller question:

   Does each recursive call to the function involve a smaller case of the original problem, leading inescapably to the base case?

3. The General-case question:

   Assuming that the recursive call(s) work correctly, does the whole function work correctly?

The Base-case question

Is there a nonrecursive way out of the function, and does the routine work correctly for this "base" case?

$$\binom{n}{k} = \binom{n-1}{k} + \binom{n-1}{k-1} \quad , \quad 1 < k < n$$

```
int Combinations(int n, int k)
{
int answer;
if(k == 1)  /*base case 1*/
  return n;
 else if (n == k)  /* base case 2*/
  return 1;
 else
  answer = Combinations(n-1, k) + Combinations(n-1, k-1);
 return answer;
}
```

Base cases

$$\binom{n}{1} = n \quad (k = 1), \quad \binom{n}{n} = 1 \quad (k = n)$$

The Smaller-caller question

Does each recursive call to the function involve a smaller case of the original problem, leading inescapably to the base case?

```
int Combinations(int n, int k)
{
int answer;
if(k == 1)  /*base case 1*/
  return n;
 else if (n == k)  /* base case 2*/
  return 1;
 else
  answer = Combinations(n-1, k) + Combinations(n-1, k-1);
 return answer;
}
```

$$\binom{n}{k} = \binom{n-1}{k} + \binom{n-1}{k-1} \quad , \quad 1 < k < n$$

Base cases

$$\binom{n}{1} = n \quad (k = 1), \quad \binom{n}{n} = 1 \quad (k = n)$$

The General-case question

Assuming that the recursive call(s) work correctly, does the whole function work correctly?

```
int Combinations(int n, int k)
{
int answer;
if(k == 1)  /*base case 1*/
  return n;
 else if (n == k)  /* base case 2*/
  return 1;
 else
  answer = Combinations(n-1, k) + Combinations(n-1, k-1);
 return answer;
}
```

$$\binom{n}{k} = \binom{n-1}{k} + \binom{n-1}{k-1} \quad , \quad 1 < k < n$$

Base cases

$$\binom{n}{1} = n \quad (k = 1), \binom{n}{n} = 1 \quad (k = n)$$

Recursive Fibonacci Function

- The Fibonacci Sequence (菲波拿契數列) is given as 1,1,2,3,5,8,13,21,…..
- The n-th Fibonacci number is given as follows.
- The Recursive Solution:

F(n)=F(n-1) + F(n-2) for n >2

With base case

F(1)=1; F(2)=1

Recursive Fibonacci Function

- Exercise: Write the recursive function code for the Fibonacci function to compute the n-th Fibonacci number.

Code:

```
int Fibonacci( int n)
{
    .........
}
```

Recursive Fibonacci Function

- Exercise: Write a recursive function code for the Fibonacci function to compute the n-th Fibonacci number.

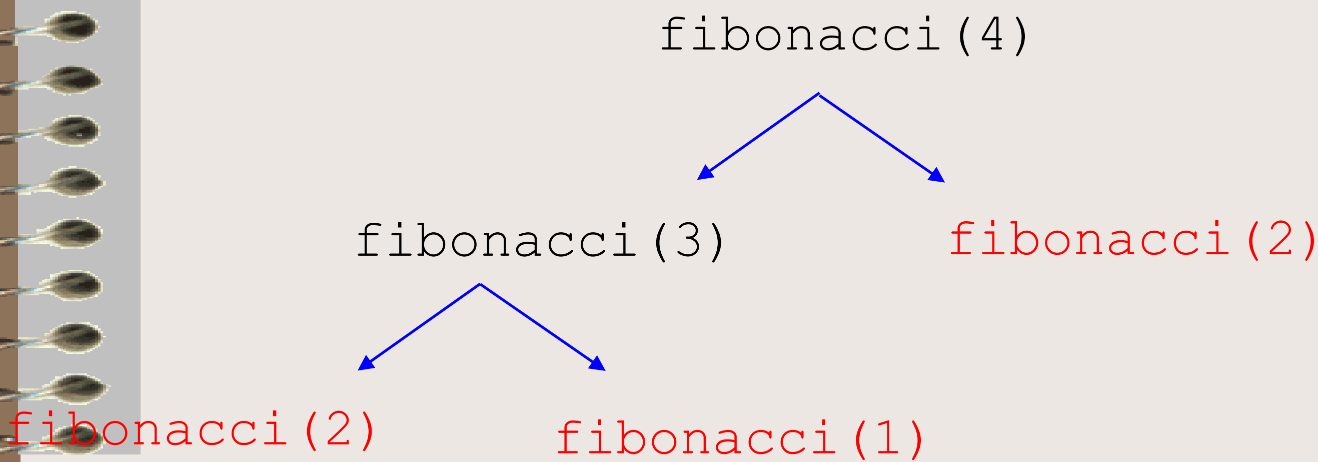
int fibonacci(int n)

```
{
        int ans;

        if (n == 1  ||  n == 2)
                ans = 1;
        else
                ans = fibonacci(n - 2)  +  fibonacci(n - 1);

        return (ans);
}
```

Execution Trace (decomposition)

```
          fibonacci(4)

fibonacci(3)          fibonacci(2)
```

Execution Trace (decomposition)



Execution Trace (composition)

fibonacci(4)

+

fibonacci(3) fibonacci(2)

+

fibonacci(2)->1 fibonacci(1)->1

Execution Trace (composition)

```
            fibonacci(4)
                 +
fibonacci(3)->2      fibonacci(2)->1
```

Execution Trace (composition)

```
fibonacci(4)->3
```

Recursion versus Iteration

- Repetition
  - Iteration: explicit loop
  - Recursion: repeated function calls
- Termination
  - Iteration: loop condition fails
  - Recursion: base case recognized
- Both can have infinite loops
  - Be careful about the termination condition