程序设计
Programming

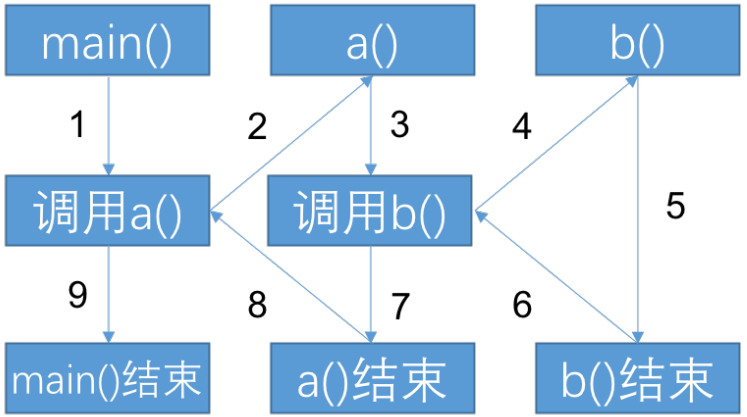
Lecture 9: 函数与程序结构

1. 函数的嵌套调用
2. 递归函数
3. 编译预处理
4. 创建多文件Project
   - ✓文件模块间的依赖和通信

1、函数的嵌套调用

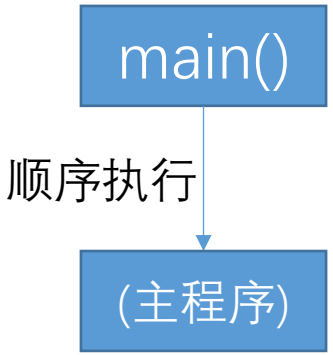
函数的嵌套调用

- 主函数可以调用自定义函数
- 自定义函数可以调用其他自定义函数
  - ✓自定义函数间可以互相调用



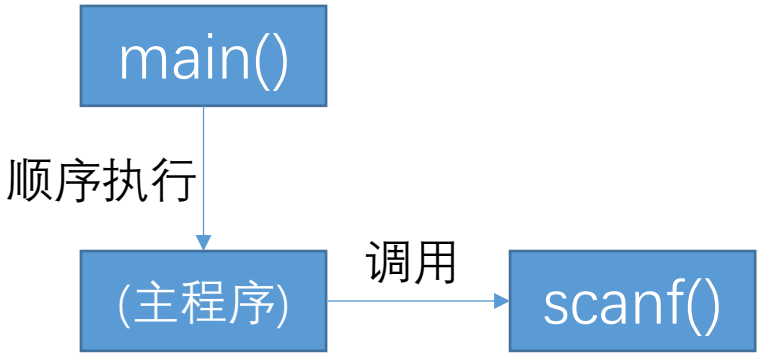
函数的嵌套调用

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


int main()
{
    int e, x;
    scanf("%d%d",&e,&x);
    printf("%f", pow(e, x));

    return 0;
}
```



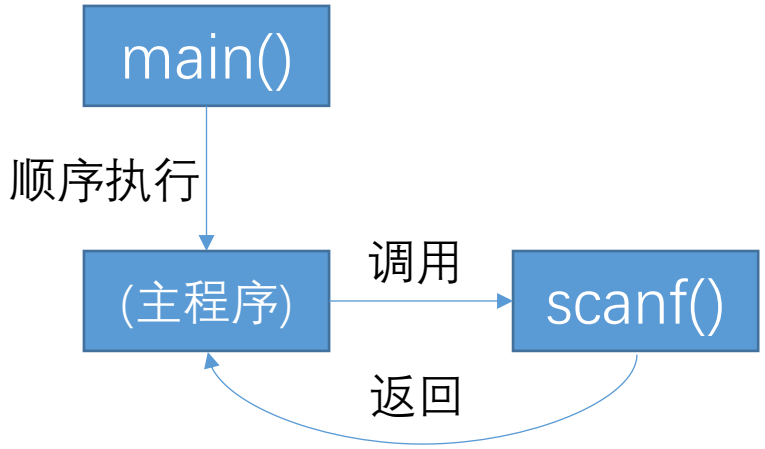
函数的嵌套调用

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


int main()
{
    int e, x;
    scanf("%d%d",&e,&x);
    printf("%f", pow(e, x));

    return 0;
}
```



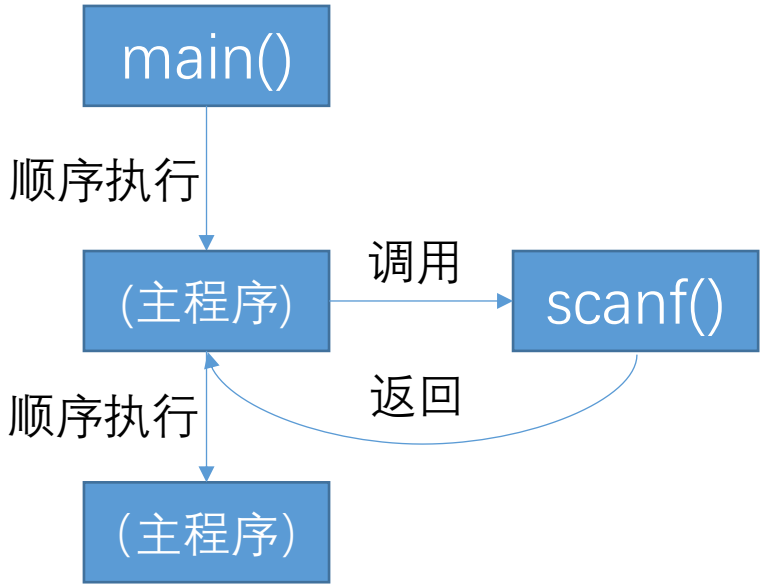
函数的嵌套调用

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


int main()
{
    int e, x;
    scanf("%d%d",&e,&x);
    printf("%f", pow(e, x));

    return 0;
}
```



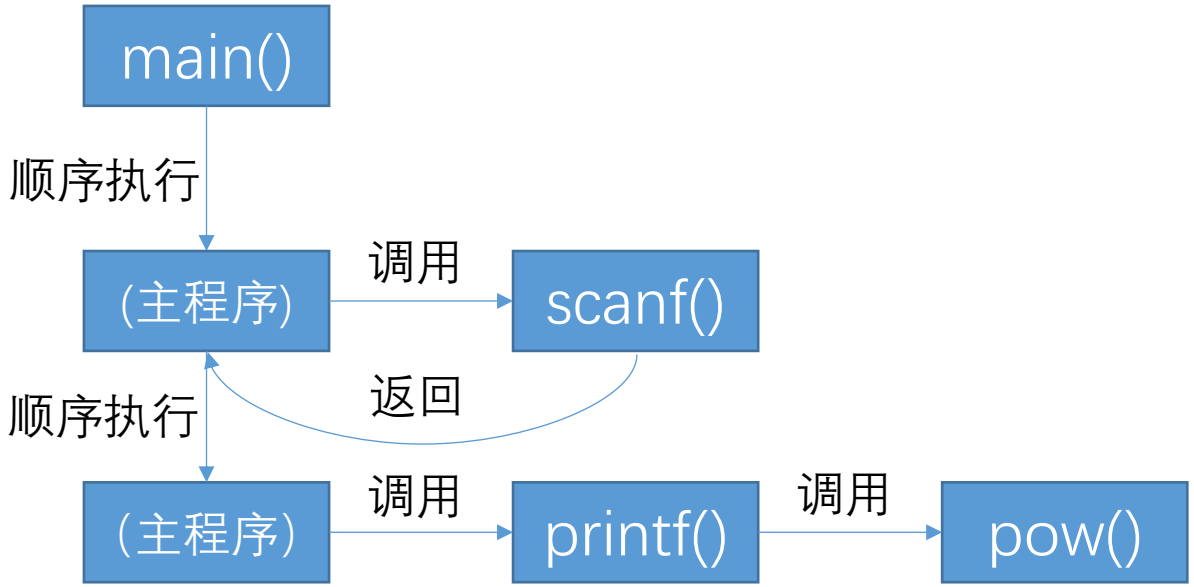
函数的嵌套调用

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


int main()
{
    int e, x;
    scanf("%d%d",&e,&x);
    printf("%f", pow(e, x));

    return 0;
}
```



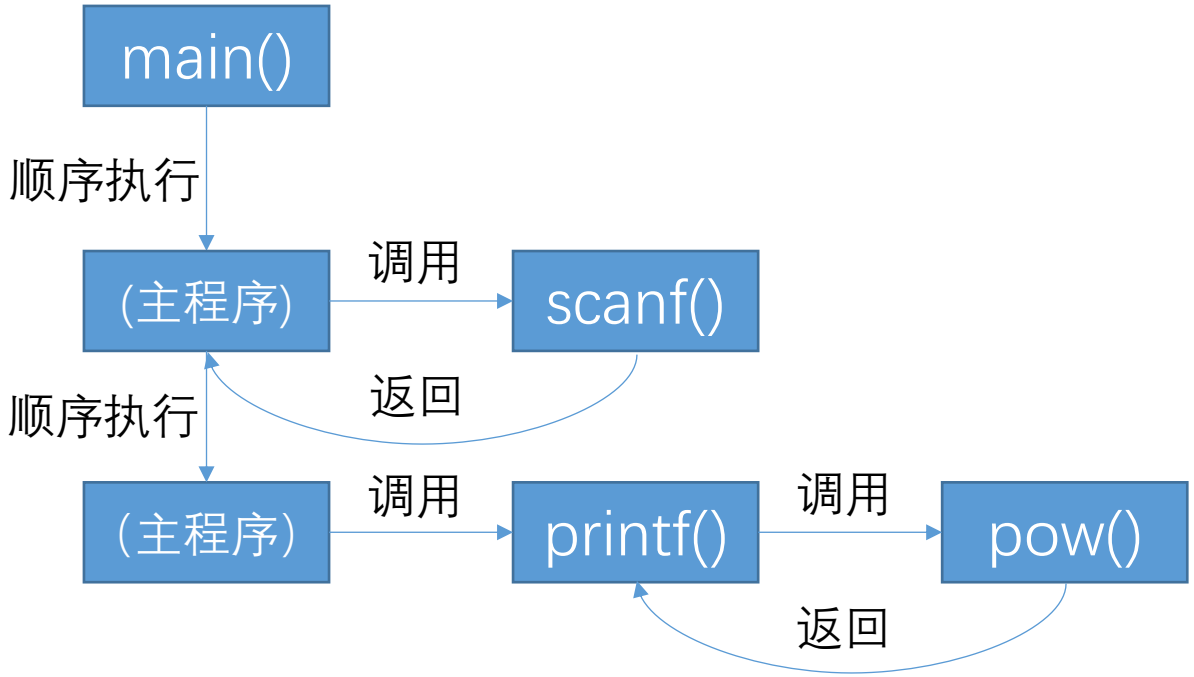
函数的嵌套调用

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


int main()
{
    int e, x;
    scanf("%d%d",&e,&x);
    printf("%f", pow(e, x));

    return 0;
}
```



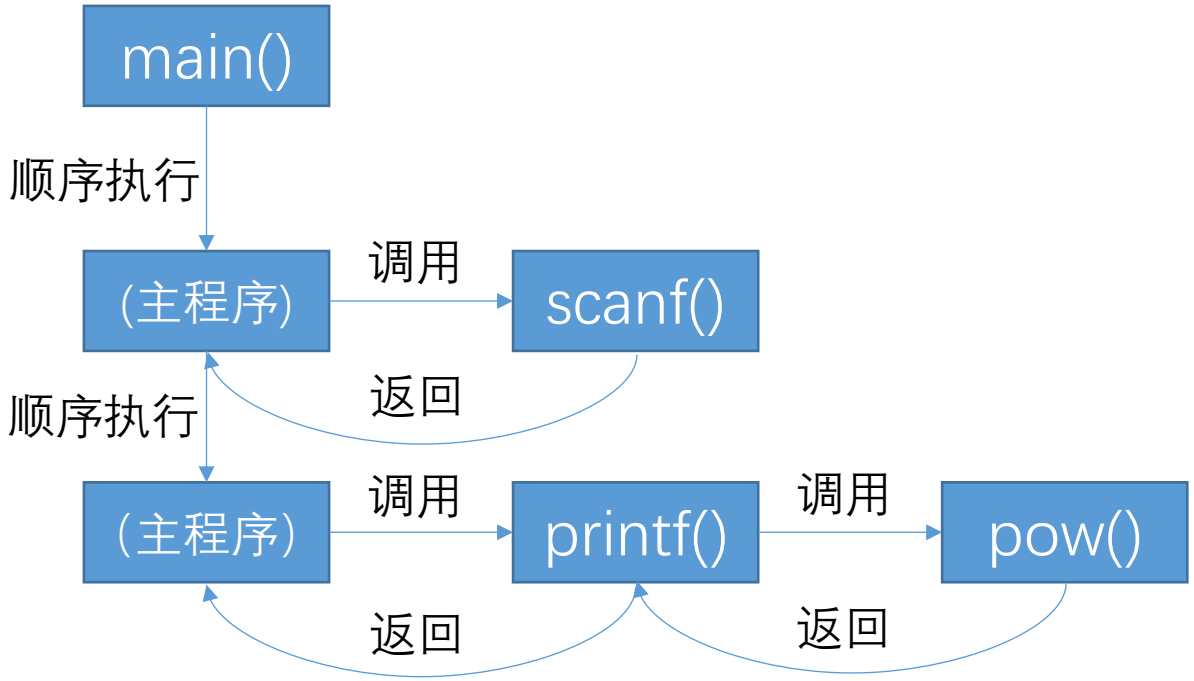
函数的嵌套调用

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


int main()
{
    int e, x;
    scanf("%d%d",&e,&x);
    printf("%f", pow(e, x));

    return 0;
}
```



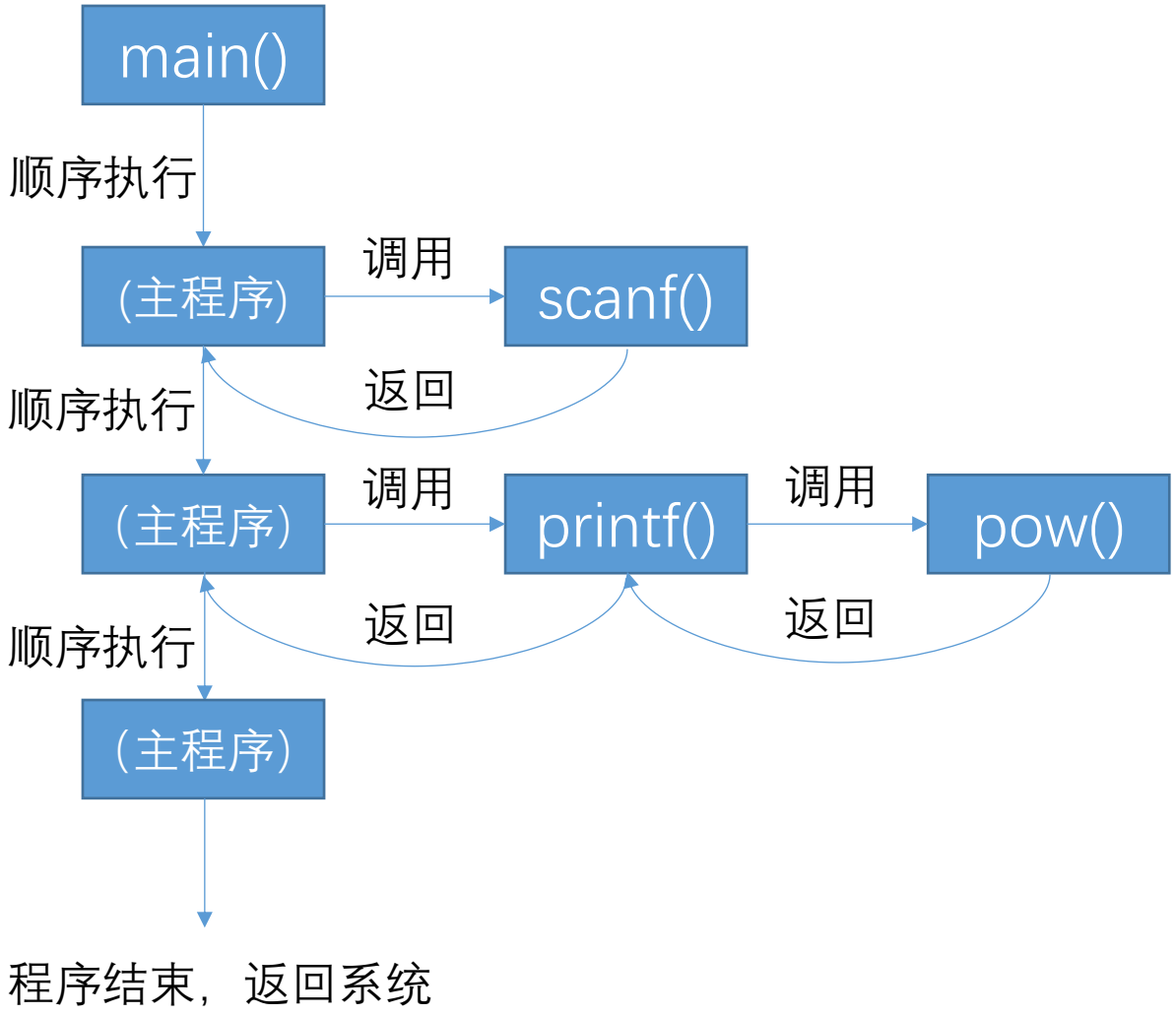
函数的嵌套调用

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


int main()
{
    int e, x;
    scanf("%d%d",&e,&x);
    printf("%f", pow(e, x));

    return 0;
}
```



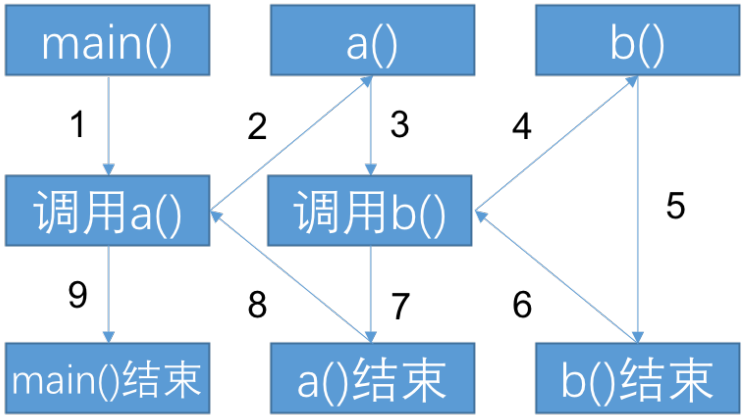
函数的嵌套调用

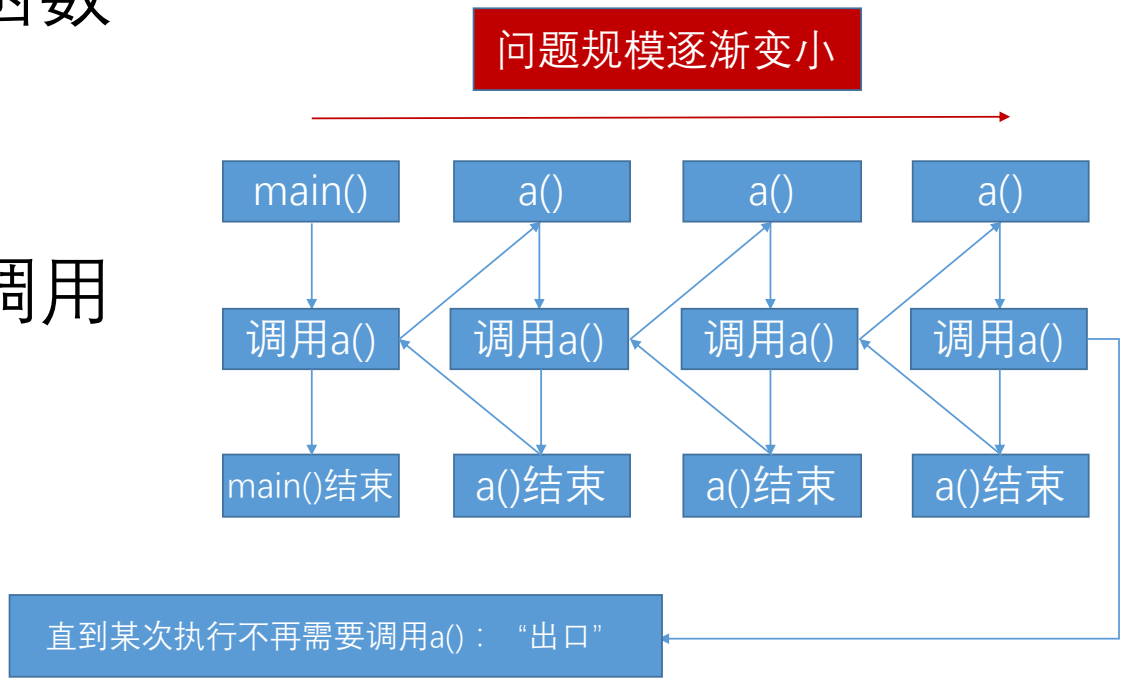
```c
#include <stdio.h>
#include <math.h>


int main()
{
    int e, x;
    scanf("%d%d",&e,&x);
    printf("%f", pow(e, x));

    return 0;
}
```



函数的嵌套调用

- 主函数可以调用自定义函数
- 自定义函数可以调用其他自定义函数
  - ✓自定义函数间可以互相调用
- 递归函数其实是一种特殊的嵌套调用
  - ✓调用自己





程序函数组织方法

- 自顶向下
  - ✓先考虑总体步骤，再考虑步骤的细节
- 逐步求精
  - ✓将复杂问题的大步骤分解为小步骤
- 函数实现
  - ✓把最终的小问题用函数来实现（必要时使用递归）

2、递归函数

递归函数

- 自我调用的函数称为递归函数（Recursive Function）
  - ✓是分治算法（divide-and-conquer）的一种代码体现
  - ✓先将大问题分解为小问题，用相同的方法分别解决这些小问题
  - ✓最后从递归出口开始，将小问题的结果逐层返回给大问题
- 在解决复杂问题时，代码实现会非常简洁高效
- 缺点：运行效率相对较低
  - ✓大量的函数调用
  - ✓可能有大量重复计算过程

递归：阶乘

- fact(n)：求n!

fact(n) = fact(n-1) * n

fact(n-1) = fact(n-2) * (n-1)

fact(n-2) = fact(n-3) * (n-2)

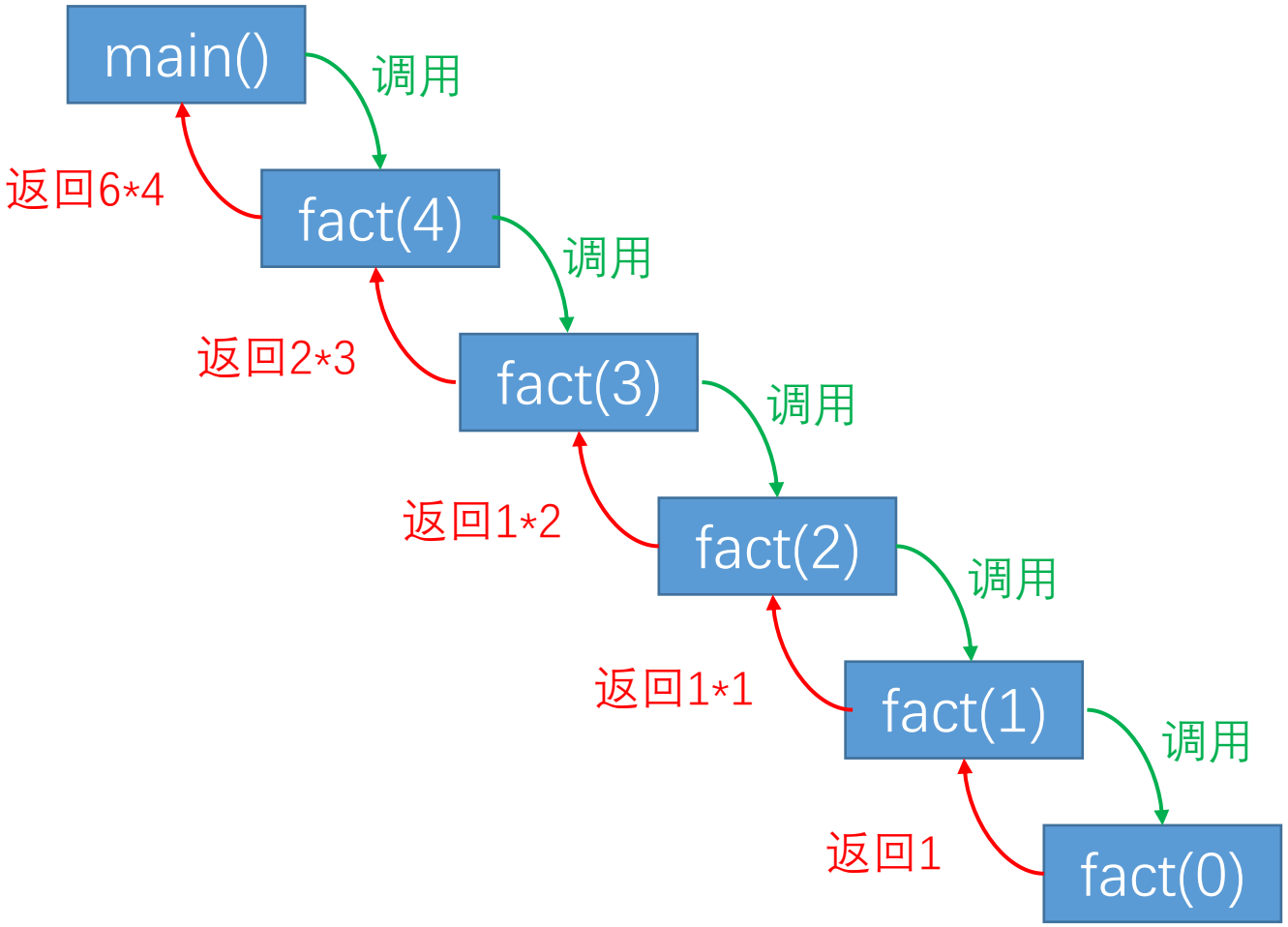
.....

.....

fact(0) = 1

```
3   int fact(int n)
4   {
5       if (n == 0)
6           return 1;
7       return fact(n-1) * n;
8   }
9
10
11  int main()
12  {
13      int n = 4;
14      printf("%d", fact(n));
15      return 0;
16  }
```

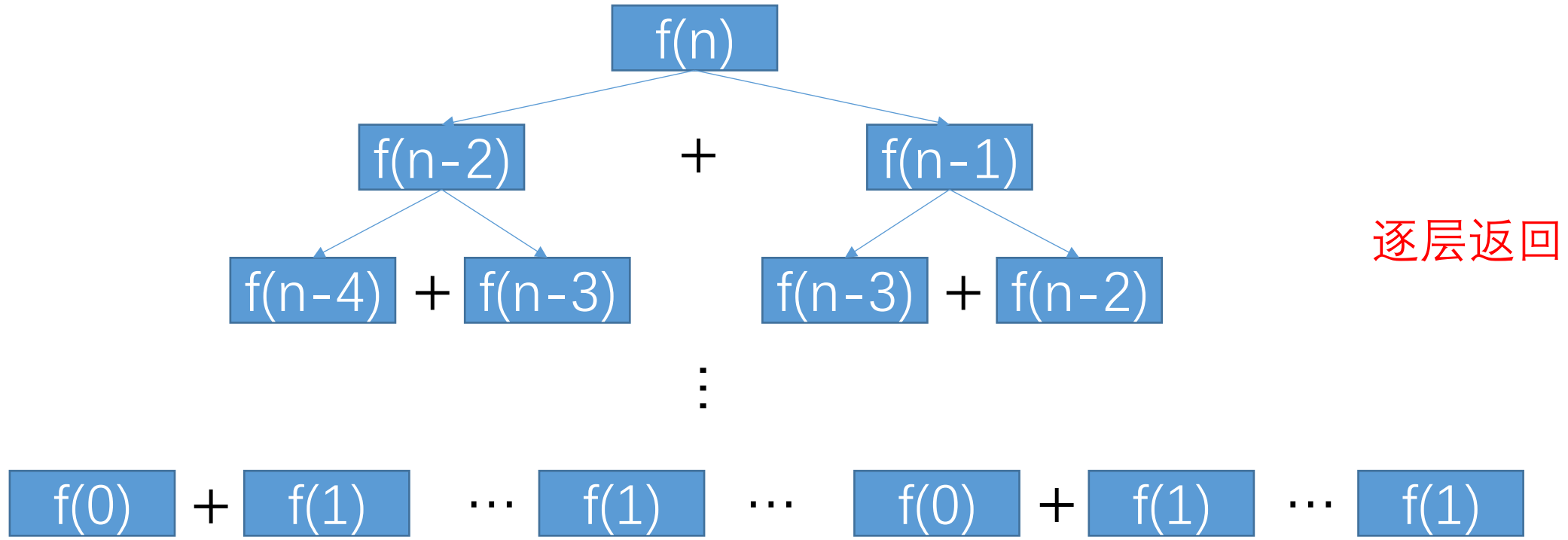
递归：阶乘




```
3   int fact(int n)
4   {
5       if (n == 0)
6           return 1;
7       return fact(n-1) * n;
8   }
9
10
11  int main()
12  {
13      int n = 4;
14      printf("%d", fact(n));
15      return 0;
16  }
```

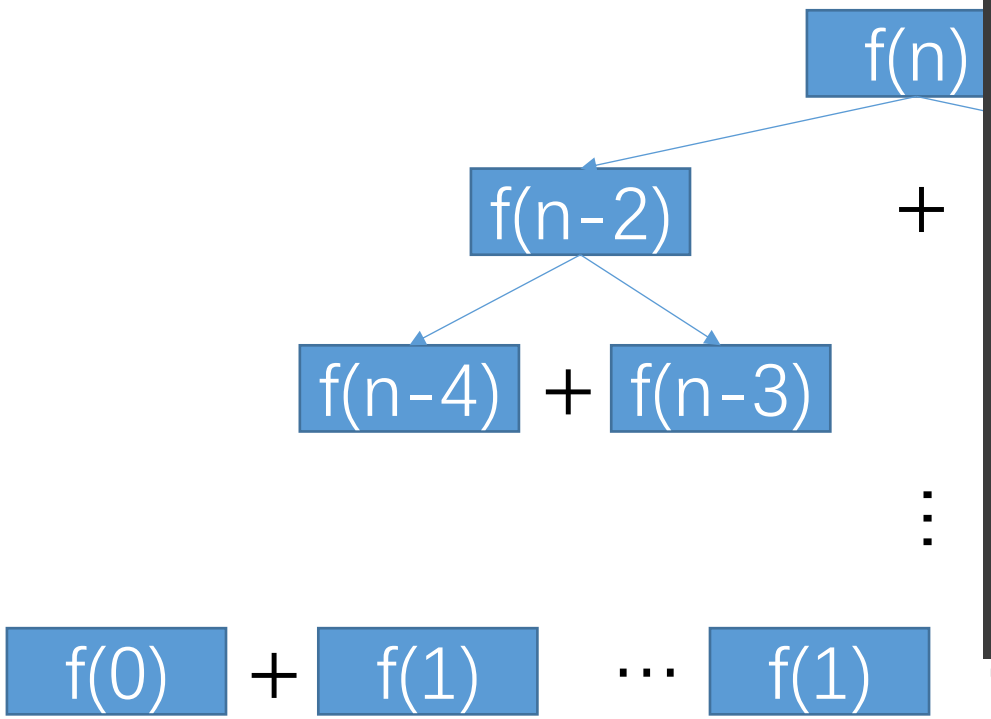
递归：斐波那契数列

- 又称黄金分割数列：1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, ...
- f(1)=1, f(2)=1, f(n)=f(n-2)+f(n-1), for n>2



递归：斐波那契数列

- 又称黄金分割数列：1, 1, 2, 3, 5,
- f(1)=1, f(2)=1, f(n)=f(n-2)+f(n-1),



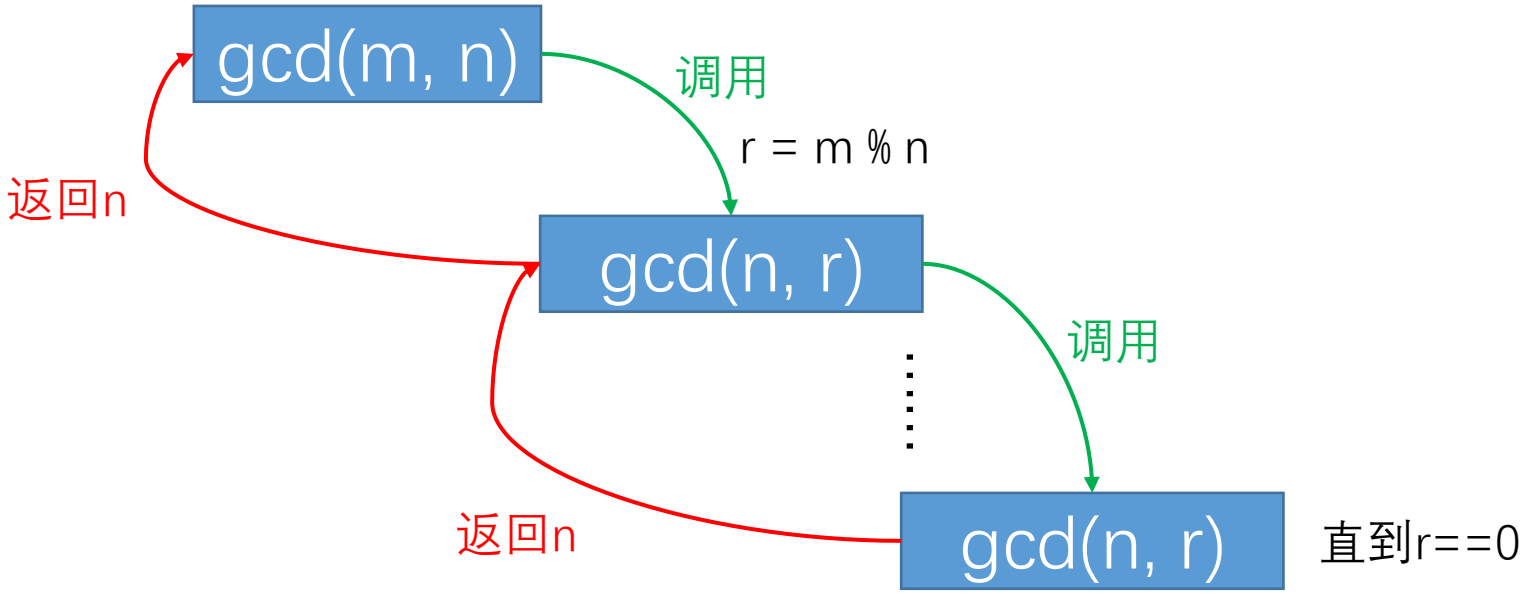

```c
#include <stdio.h>

int fib(int n)
{
    if (n == 1 || n == 2)
        return 1;
    else
        return fib(n-1) + fib(n-2);
}

int main()
{
    int a = 10;
    printf("%d", fib(a));
    return 0;
}
```

递归：最大公约数

- 求两个正整数m和n的最大公约数gcd（greatest common divisor）

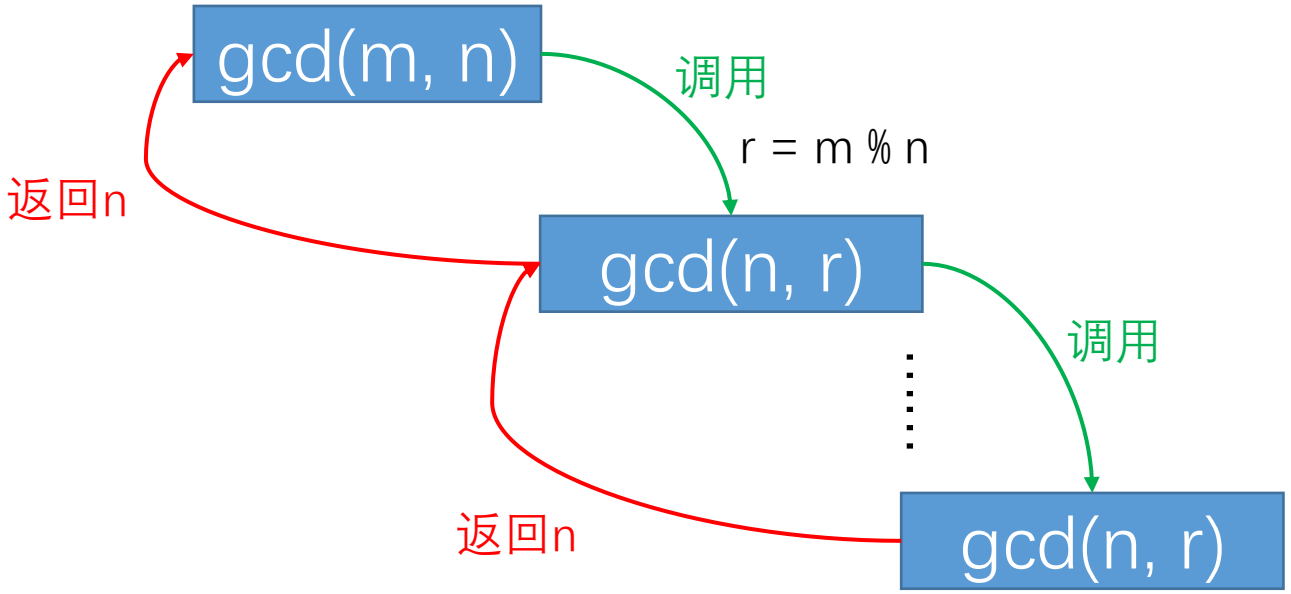
递归：最大公约数

- 求两个正整数m和n的最大公约数gcd（greatest common divisor）
  - 辗转相除法：gcd(m, n) == gcd(m%n, n)，假设m>=n
  - 即两数的gcd等于两数相除（大除以小）余数和较小数的gcd



递归：最大公约数

- 求两个正整数m和n的最大公约数gcd（
  - 辗转相除法：gcd(m, n) == gcd(m%n, n),
  - 即两数的gcd等于两数相除（大除以小）余

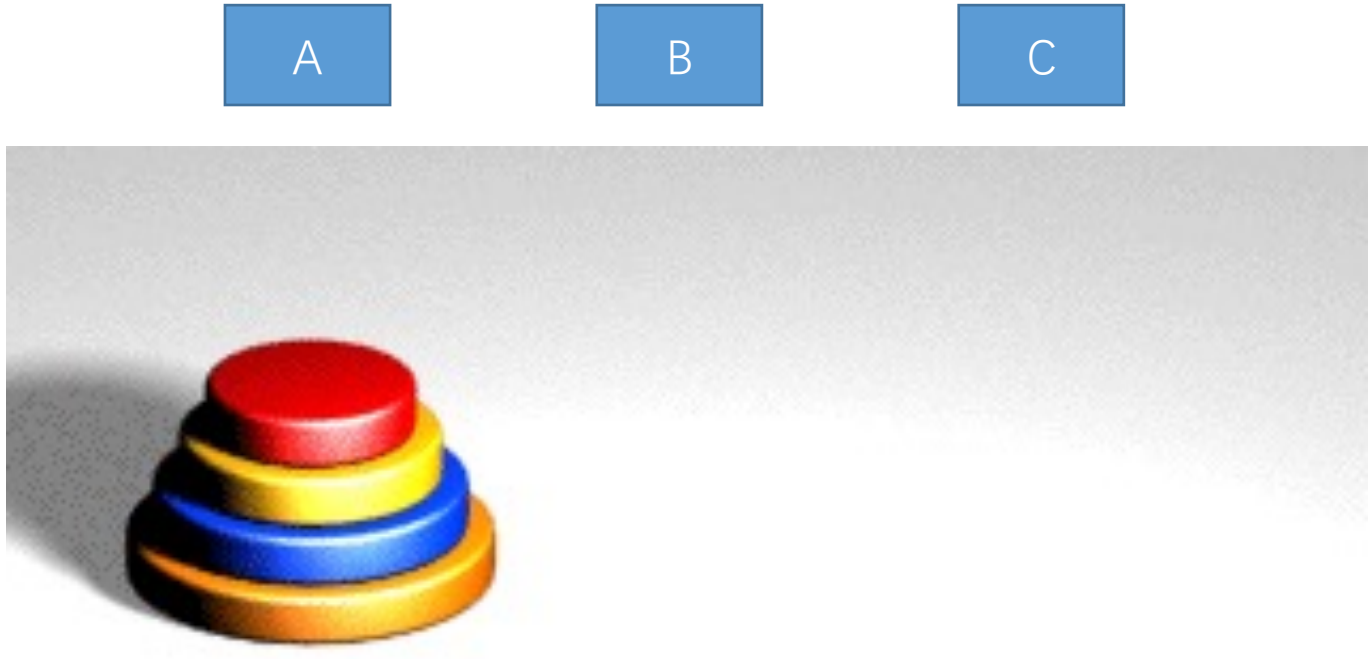



```
3  int gcd(int m, int n)
4  {
5      if(m < n)
6      {
7          int tmp = m;
8          m = n;
9          n = tmp;
10     }
11     if(n == 0)
12         return m;
13     return gcd(m % n, n);
14 }
```

直到r==0

递归：汉诺塔

- Tower of Hanoi



递归：汉诺塔

- Tower of Hanoi
  - ✓有ABC三个点，将所有盘子从A搬到C
  - ✓一次只能搬运一个盘子，放在另外两个点之一
  - ✓大盘永远只能在小盘下面

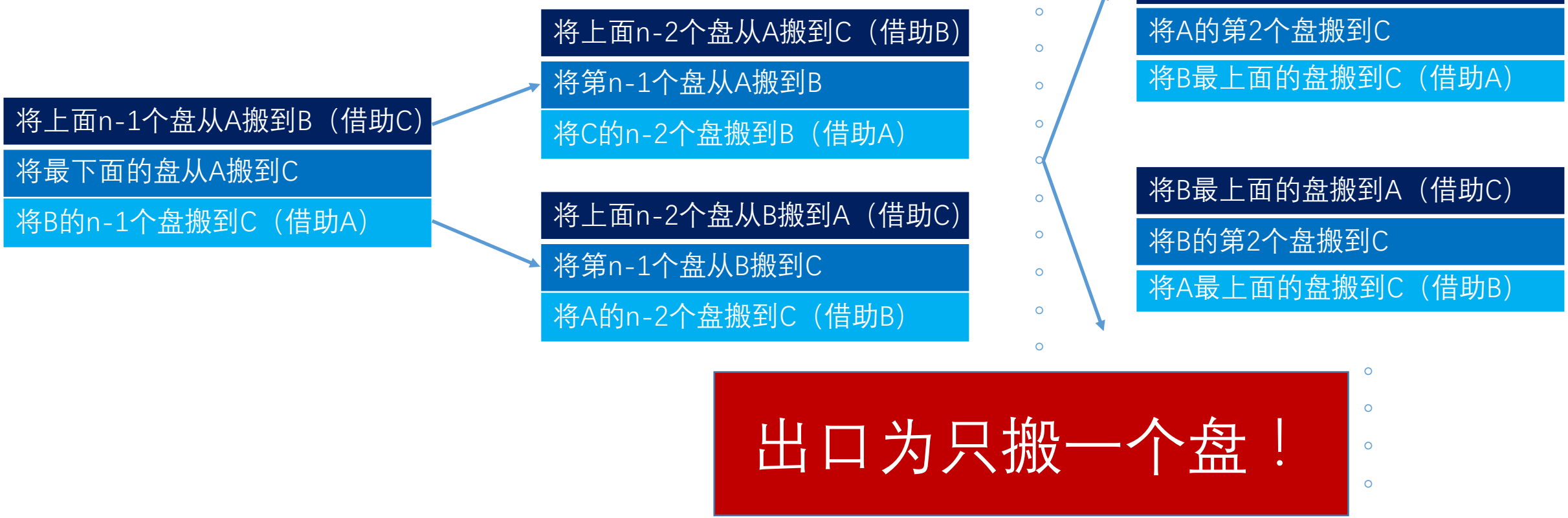
递归：汉诺塔

- Tower of Hanoi
  - ✓有ABC三个点，将所有盘子从A搬到C
  - ✓一次只能搬运一个盘子，放在另外两个点之一
  - ✓大盘永远只能在小盘下面

- 递归思想：将n个盘子从A搬到C（借助B）的子问题
  - ✓将上面n-1个盘子从A搬到B（借助C）
  - ✓将最下面的盘子从A搬到C
  - ✓将B的n-1个盘子搬到C（借助A）

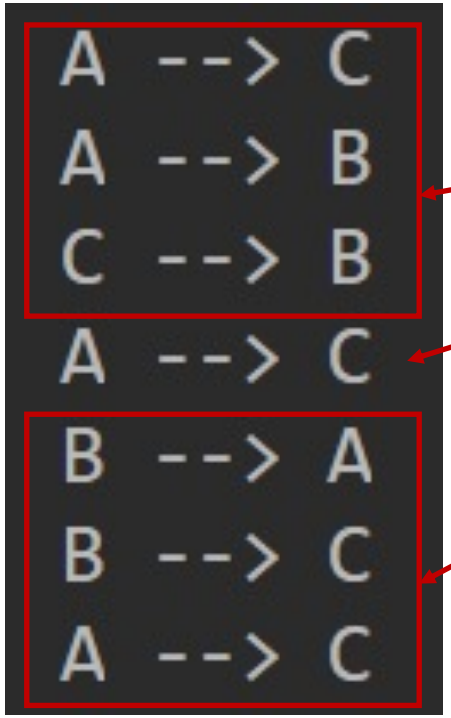
递归：汉诺塔

- Tower of Hanoi



递归：汉诺塔

- Tower of Hanoi




```c
int hanoi(int n, char A, char B, char C)
{
    if (n == 1)
    {
        printf("%c --> %c\n", A, C);
        return 0;
    }
    hanoi(n-1, A, C, B);
    printf("%c --> %c\n", A, C);
    hanoi(n-1, B, A, C);

}

int main()
{
    int n = 3;
    hanoi(n, 'A', 'B', 'C');
    return 0;
}
```

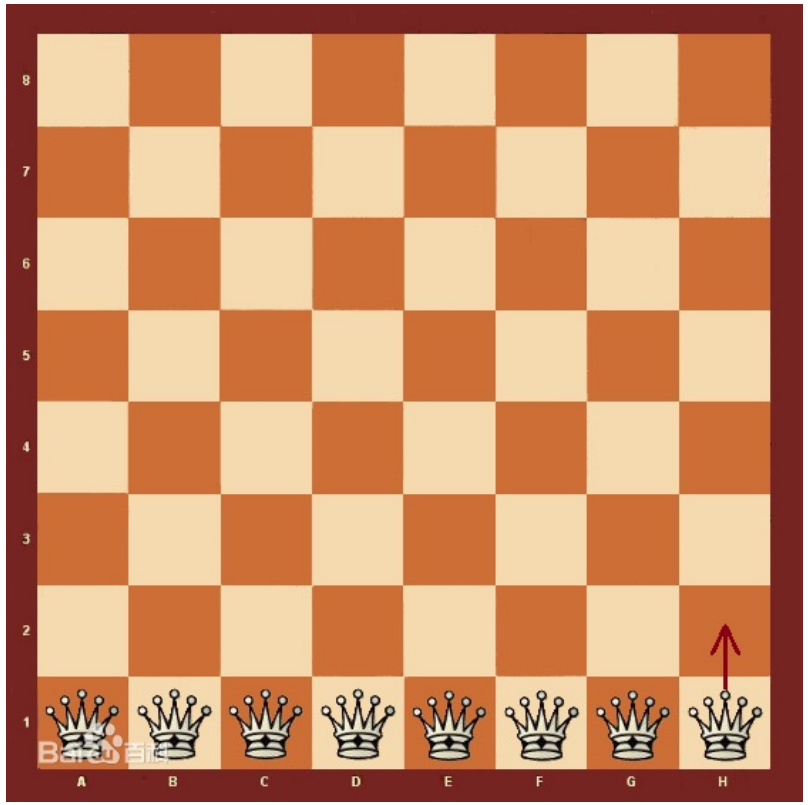
递归：八皇后

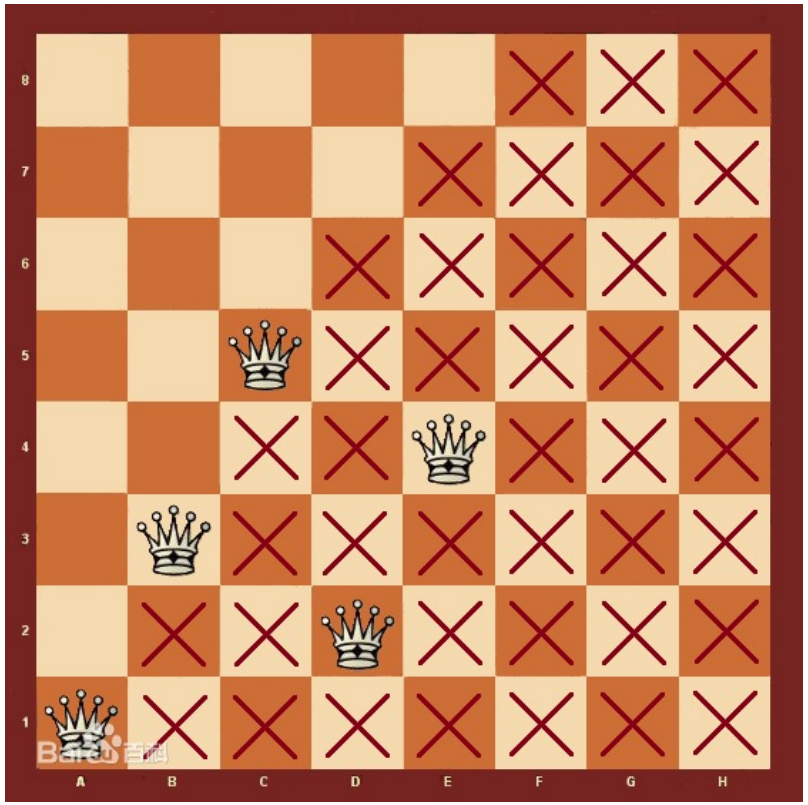
- 八皇后问题（英文：Eight queens），是由国际西洋棋棋手马克斯·贝瑟尔于1848年提出的问题，是回溯算法的典型案例。（可以用递归实现）

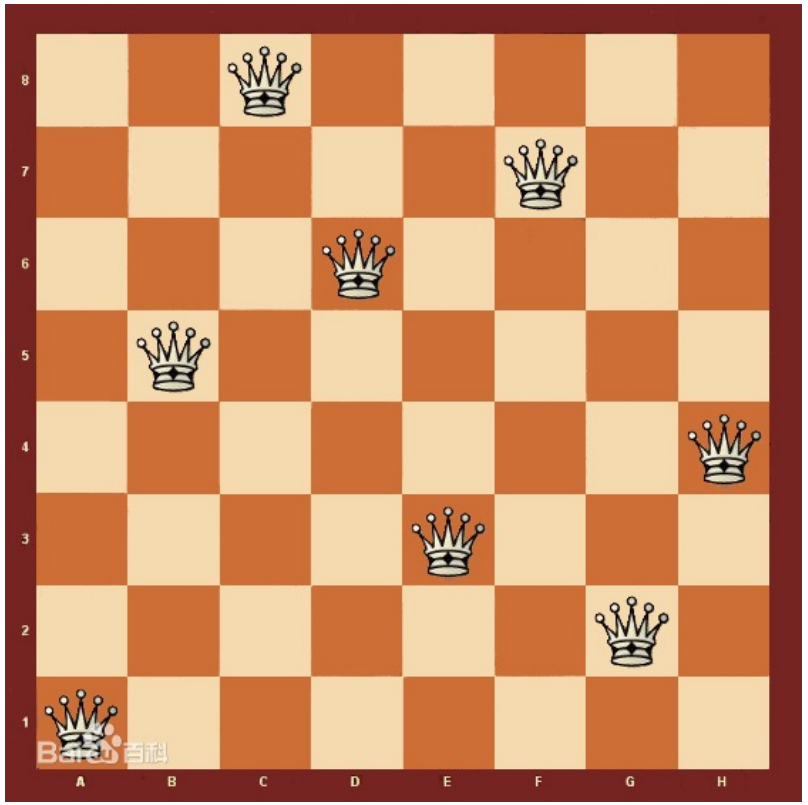
- 问题表述为：在8×8格的国际象棋上摆放8个皇后，使其不能互相攻击，即任意两个皇后都不能处于同一行、同一列或同一斜线上，问有多少种摆法。

- 高斯认为有76种方案。1854年在柏林的象棋杂志上不同的作者发表了40种不同的解，后来有人用图论的方法解出92种结果。如果经过±90度、±180度旋转，和对角线对称变换的摆法看成一类，共有42类。计算机发明后，有多种计算机语言可以编程解决此问题。

递归：八皇后







更多递归问题：https://www.geeksforgeeks.org/recursion-practice-problems-solutions/

3、编译预处理

编译预处理：宏定义#define

- 定义符号常量，增加程序的可读性

  #define 宏名 宏定义字符串　　//行尾没有分号

编译预处理：宏定义#define

- 定义符号常量，增加程序的可读性

  #define 宏名 宏定义字符串　　//行尾没有分号

- 宏名是一个标识符，一般将所有字母都大写
- 宏定义字符串是宏的实现过程，可以是浮点型、整型或字符串

  #define PI 3.1415

  #define TRUE 1

  #define UNIVERSITY “East China Normal University”

编译预处理：宏定义#define

- 定义符号常量，增加程序的可读性

  #define 宏名 宏定义字符串　　//行尾没有分号

- 宏名是一个标识符，一般将所有字母都大写
- 宏定义字符串是宏的实现过程，可以是浮点型、整型或字符串

  #define PI 3.1415

  #define TRUE 1

  #define UNIVERSITY “East China Normal University”

- 程序中出现宏名的地方，用相应的宏定义字符串直接替换

编译预处理：宏定义#define

- 宏定义的常用场景？

编译预处理：宏定义#define

- 宏定义的常用场景？
  - ✓将常量表示成符号，增加程序灵活性
  - ✓简单的函数功能（宏名可以带参数，不要忘记括号的使用！！）
  - ✓多次使用的相同长字符串

编译预处理：宏定义#define

- 宏定义的常用场景？
  - ✓将常量表示成符号，增加程序灵活性
  - ✓简单的函数功能（宏名可以带参数，不要忘记括号的使用！！）
  - ✓多次使用的相同长字符串

```
#define  MAX(x, y)  x > y ? x : y
#define  OPEN_ERROR    “Error opening the file: No such file or directory”
```

编译预处理：宏定义#define

- 宏定义函数和普通函数的区别
  - ✓宏定义函数在预处理时替换掉程序中的宏，程序执行时宏已经消失；普通函数在程序执行时才发生传参，主函数暂停并调用普通函数
  - ✓如果实参是表达式，宏定义函数在替换时不对表达式做计算，而是直接替换；普通函数会先计算表达式的值，再传递参数（宏定义中必要的括号很重要！！）

编译预处理：宏定义#define

- 宏定义函数和普通函数的区别
  - ✓宏定义函数在预处理时替换掉程序中的宏，程序执行时宏已经消失；普通函数在程序执行时才发生传参，主函数暂停并调用普通函数
  - ✓如果实参是表达式，宏定义函数在替换时不对表达式做计算，而是直接替换；普通函数会先计算表达式的值，再传递参数（宏定义中必要的括号很重要！！）

```
#define  SQR(x)   X*X
z = SQR(x+y)
```

??

宏定义#define和常量const的区别

- 类型和安全检查不同
  - ✓宏定义（macro definition）是字符替换，编译阶段没有类型安全检查，可能产生意想不到错误；
  - ✓const常量是变量被修饰为常量，有类型区别，在编译阶段进行类型检查
- 编译器处理不同
  - ✓宏定义是一个“编译时”概念，在预处理阶段展开，不能对宏定义进行调试，生命周期结束于编译时期；
  - ✓const常量是一个"运行时"概念，在程序运行使用
- 存储方式不同
  - ✓宏定义是字符直接替换，存储于程序的代码段中；
  - ✓const常量需要进行内存分配，存储于程序的数据段中

宏定义#define和常量const的区别

- 作用域不同
  - ✓宏定义可以在程序中任何地方被使用，也可以通过#include在其他程序中被使用
  - ✓const常量只在其被声明的定义域内有效
- 定义后能否取消
  - ✓宏定义可以通过#undef来使之前的宏定义失效
  - ✓const常量定义后将在定义域内永久有效且不可重新赋值

编译预处理：头文件包含#include

- 将指定的模块插入到当前文件中
  - ✓模块的头文件中一般包含特定模块的宏、变量或函数定义
  - ✓定义之后可以在其他文件中通过#include该头文件使用这些宏、变量或者函数，实现模块化编程，避免重复工作，方便项目合作

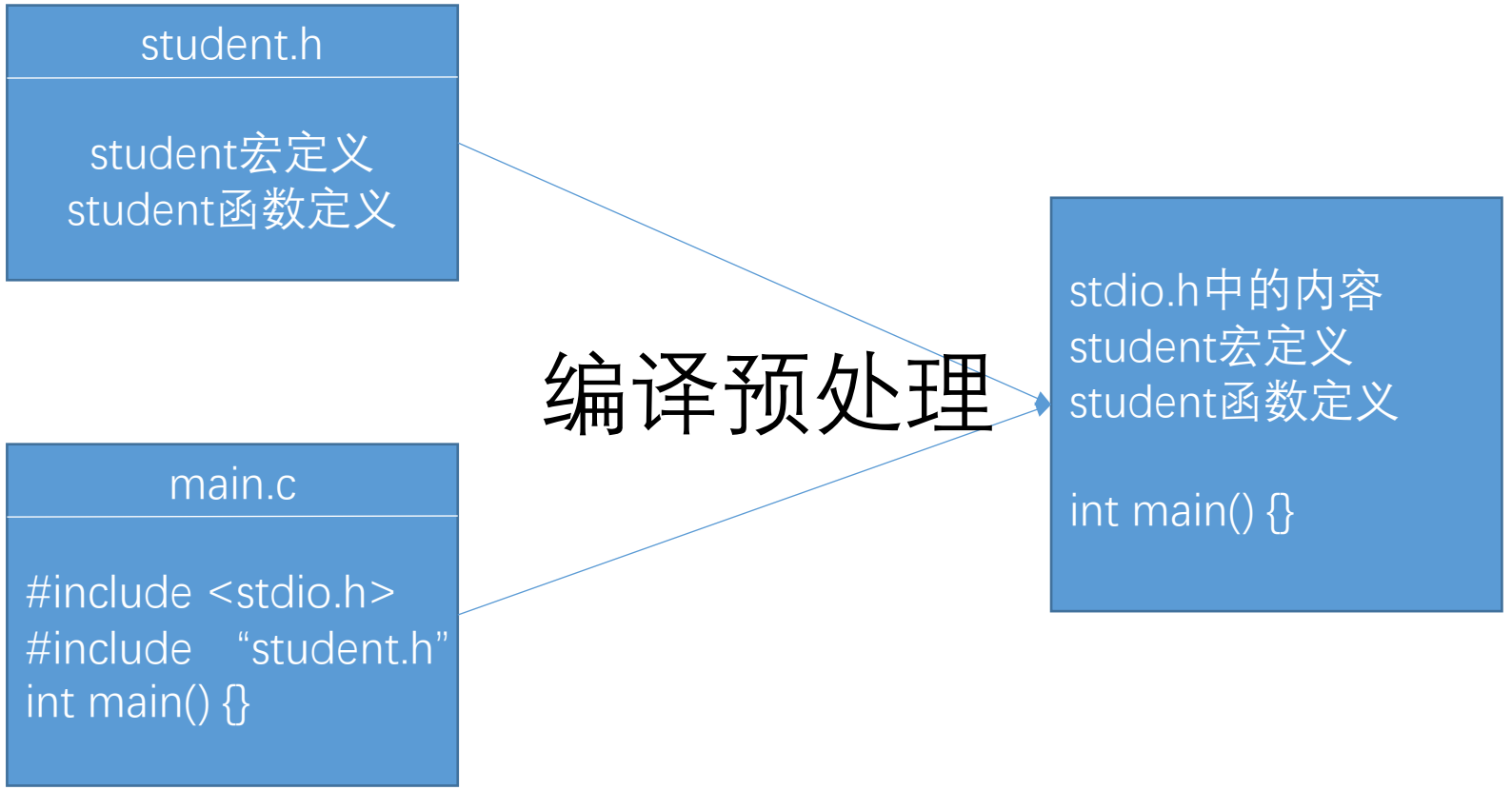
编译预处理：头文件包含#include

- 将指定的模块插入到当前文件中
  - ✓模块的头文件中一般包含特定模块的宏、变量或函数定义
  - ✓定义之后可以在其他文件中通过#include该头文件使用这些宏、变量或者函数，实现模块化编程，避免重复工作，方便项目合作

- 头文件分为标准头文件和自定义头文件

```
#include <标准头文件>          //直接到系统路径查找相应的头文件
#include  “自定义头文件”       //先查找当前工作路径，再查找系统路径
```

编译预处理：头文件包含#include



编·　　引含#include

C 标准库头文件

<assert.h>	条件编译宏，将参数与零比较
<complex.h> (C99 起)	复数运算
<ctype.h>	用来确定包含于字符数据中的类型的函数
<errno.h>	报告错误条件的宏
<fenv.h> (C99 起)	浮点数环境
<float.h>	浮点数类型的极限
<inttypes.h> (C99 起)	整数类型的格式转换
<iso646.h> (C95 起)	符号的替代写法
<limits.h>	基本类型的大小
<locale.h>	本地化工具
<math.h>	常用数学函数
<setjmp.h>	非局部跳转
<signal.h>	信号处理
<stdalign.h> (C11 起)	alignas 与 alignof 便利宏
<stdarg.h>	可变参数
<stdatomic.h> (C11 起)	原子类型
<stdbool.h> (C99 起)	布尔类型
<stddef.h>	常用宏定义
<stdint.h> (C99 起)	定宽整数类型
<stdio.h>	输入/输出
<stdlib.h>	基础工具：内存管理、程序工具、字符串转换、随机数
<stdnoreturn.h> (C11 起)	noreturn 便利宏
<string.h>	字符串处理
<tgmath.h> (C99 起)	泛型数学（包装 math.h 和 complex.h 的宏）
<threads.h> (C11 起)	线程库
<time.h>	时间/日期工具
<uchar.h> (C11 起)	UTF-16 和 UTF-32 字符工具
<wchar.h> (C95 起)	扩展多字节和宽字符工具
<wctype.h> (C95 起)	用来确定包含于宽字符数据中的类型的函数

https://zh.cppreference.com/w/c/header

编译预处理：条件编译 #if

- 将代码中的部分语句进行编译，使得编译后的目标代码更加精简

```
#define  FLAG  1
int main()
{
#if  FLAG
     程序段  1
#else
     程序段  2
#endif
}
```

```
#define  FLAG  1
int main()
{
#if  FLAG < 1
     程序段  1
#elif FLAG == 1
     程序段  2
#else
     程序段  3
#endif
}
```

编译预处理：条件编译 #if

- 将代码中的[illegible]代码更加精简

```
#define  FL
int main()
{
#if  FLAG
    程序段
#else
    程序段
#endif
}
```

```
1  #include <stdio.h>
2  #define MAX 100
3  int main()
4  {
5  #if MAX < 100
6      printf("MAX小于100");
7  #elif MAX == 100
8      printf("MAX等于100");
9  #else
10     printf("MAX大于100");
11 #endif
12     return 0;
13 }
```

只有这两段
代码参与编译

编译预处理：条件编译 #if

- 将代码中的部分语句进行编译，使得编译后的目标代码更加精简

```
#define  FLAG  1
int main()
{
#if  FLAG
     程序段  1
#else
     程序段  2
#endif
}
```

```
#define  FLAG  1
int main()
{
#if  FLAG < 1
     程序段  1
#elif FLAG == 1
     程序段  2
#else
     程序段  3
#endif
}
```

编译预处理：条件编译 #ifdef #ifndef

- 将代码中的部分语句进行编译

```
#define  FLAG
int main()
{
#ifdef  FLAG
    程序段 1
#else
    程序段 2
#endif
}
```

```
#define  FLAG
int main()
{
#ifndef  FLAG
    程序段 1
#else
    程序段 2
#endif
}
```

编译预处理：条件编译 #ifdef #ifndef

- 将代码中

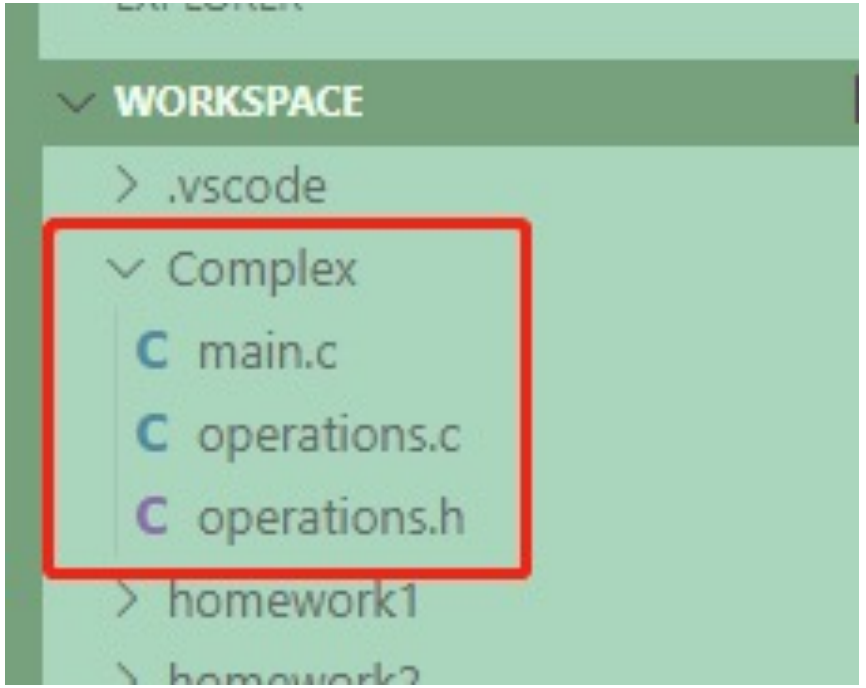
```
#define  F
int main()
{
#ifdef  FL
    程序段
#else
    程序段
#endif
}
```

```
1  #include <stdio.h>
2  #define MAX 100
3  int main()
4  {
5  #ifndef MAX
6      printf("MAX未定义");
7  #else
8      printf("MAX已定义");
9  #endif
10     return 0;
11 }
```

4、建立多文件C Project

建立多文件C Project：复数运算

1. 在VSCode中新建Project文件夹Complex，并创建三个文件：main.c, operations.h, operations.c



建立多文件C Project：复数运算

2. 编写代码并保存：operations.h，operations.c和main.c

```c
typedef struct complex
{
    int real;
    int imag;
}complex;

complex add(complex n1, complex n2);

complex sub(complex n1, complex n2);
```

```c
#include "operations.h"

complex add(complex n1, complex n2)
{
    complex res;
    res.real = n1.real + n2.real;
    res.imag = n1.imag + n2.imag;
    return res;
}

complex sub(complex n1, complex n2)
{
    complex res;
    res.real = n1.real - n2.real;
    res.imag = n1.imag - n2.imag;
    return res;
}
```

```c
#include <stdio.h>
#include "operations.h"

int main()
{
    complex n1, n2, res;
    scanf("%d%d%d%d", &n1.real,
                      &n1.imag,
                      &n2.real,
                      &n2.imag);
    res = add(n1, n2);
    printf("%d+%d\n", res.real, res.imag);
    res = sub(n1, n2);
    printf("%d+%d\n", res.real, res.imag);
    return 0;
}
```

建立多文件C Project：复数运算

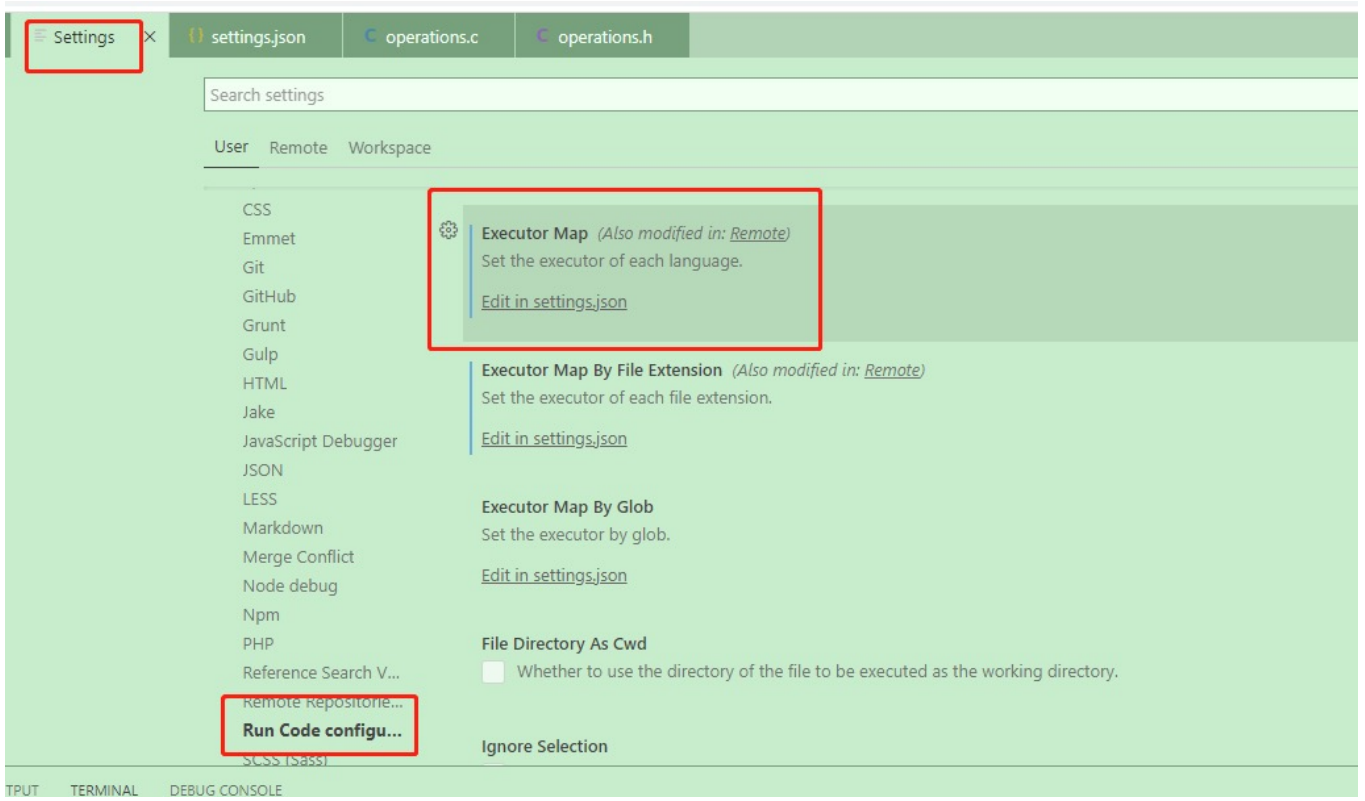
3. 使用code runner运行main.c，出现如下错误：

```
abc@c53ac3bf9413:~/workspace/Complex$ cd "/config/workspace/Com
kspace/Complex/"main
/tmp/ccUAWYqq.o: In function `main':
main.c:(.text+0x5b): undefined reference to `add'
main.c:(.text+0x8b): undefined reference to `sub'
collect2: error: ld returned 1 exit status
```

主函数找不到add和sub函数！

建立多文件C Project：复数运算

4. 修改编译配置文件，Settings→Extensions→Run Code configuration →Executor Map→Edit in settings.json



建立多文件C Project：复数运算

5. 修改c代码的编译命令，让其编译目录下所有.c文件，并保存。
（注释掉原来的命令，备用，添加一条修改后的命令）

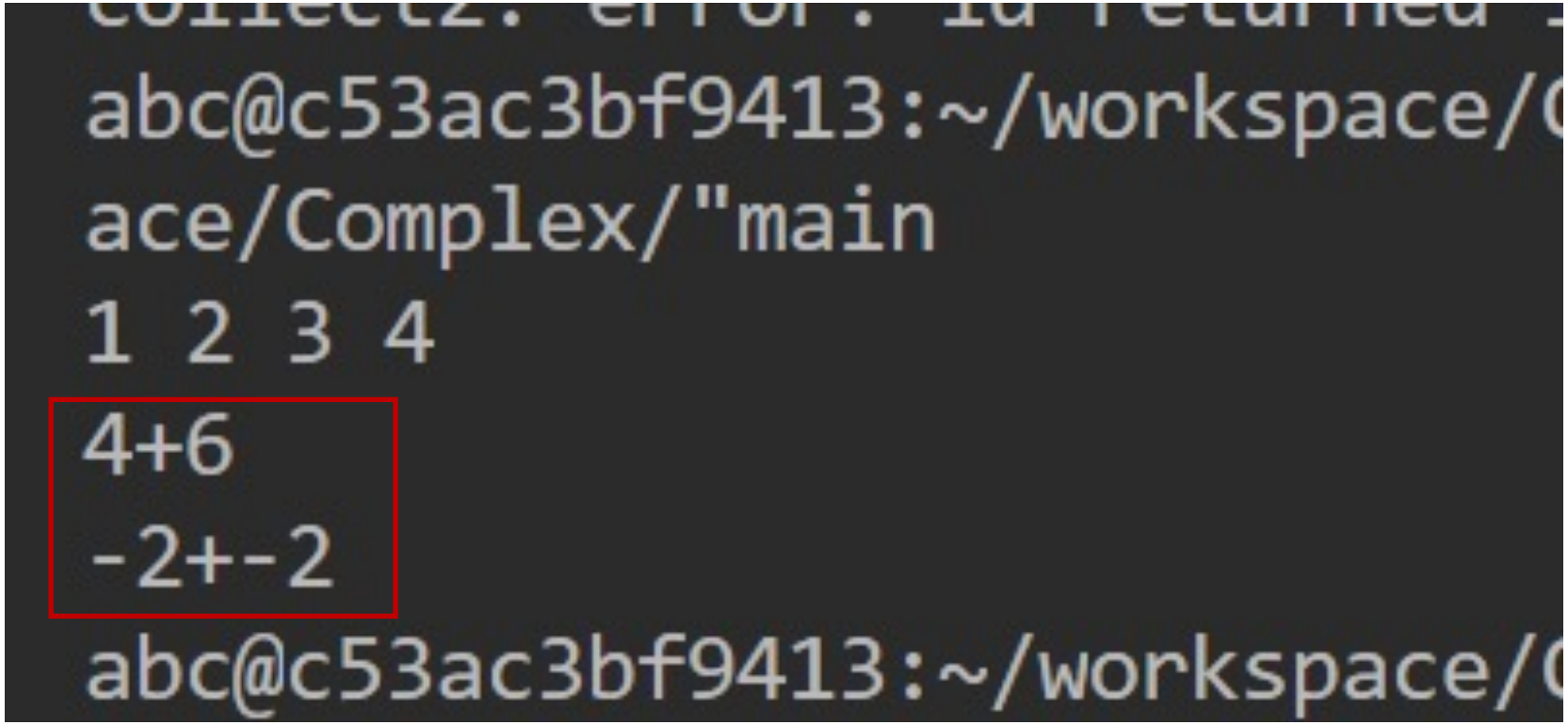
```
"terminal.integrated.fontSize": 100,
"code-runner.executorMap": {
    "javascript": "node",
    "java": "cd $dir && javac $fileName && java $fileNameWithoutExt",
    //"c": "cd $dir && gcc $fileName -o $fileNameWithoutExt -lm && $dir$fileNameWithoutExt",
    "c": "cd $dir && gcc *.c -o $fileNameWithoutExt -lm && $dir$fileNameWithoutExt",
    "cpp": "cd $dir && g++ $fileName -o $fileNameWithoutExt && $dir$fileNameWithoutExt",
    "objective-c": "cd $dir && gcc -framework Cocoa $fileName -o $fileNameWithoutExt && $dir$fileNa
    "php": "php",
    "python": "python -u",
    "perl": "perl",
    "perl6": "perl6",
    "ruby": "ruby",
    "go": "go run",
    "lua": "lua",
```

修改成

注释掉

建立多文件C Project：复数运算

6. 再次运行code runner，得到运行结果



多文件模块之间的通信：全局变量（了解）

- 外部变量（外部全局变量）
  - ✓一般会把全局变量定义在某个模块中，例如主函数所在的文件
  - ✓如果其他文件模块需要使用该全局变量，需要声明为外部变量

    extern 类型名 全局变量名;

- 静态全局变量（内部全局变量）
  - ✓全局变量原本的作用域为整个Project
  - ✓有时各个模块也会定义自己的全局变量，为了防止各文件模块的全局变量互相干扰，可以在定义时将当前模块的全局变量声明为静态类型

    static 类型名 全局变量名;
  - ✓这时该全局变量的其作用域只有当前文件模块

多文件模块之间的通信：函数（了解）

- 外部函数
  - ✓表示这个函数有可能在其他文件模块中被定义和实现
  - ✓如果当前文件模块中没有找到函数体，则去其他文件模块中寻找

    extern 类型名 函数名(参数);

- 内部函数
  - ✓为了防止各文件模块的函数互相干扰，可以在定义时将函数声明为静态类型

    static 类型名 函数名(参数);
  - ✓这时该函数的调用范围只有当前文件模块

小结

- 函数的嵌套调用
- 递归函数
  - ✓阶乘、斐波那契数列、最大公约数、汉诺塔、八皇后
- 编译预处理
  - ✓宏、包含、条件编译
- 创建多文件Project
  - ✓文件模块间的依赖和通信

L10：指针进阶