中国人民大学
RENMIN UNIVERSITY OF CHINA

第7讲 递归(1)

余力

buaayuli@ruc.edu.cn

什么是递归

- 递归是函数自己调用自己
- 要想理解递归，你要先理解递归.....
- 你做了一个梦，在梦里你做了一个梦，在梦里的梦里，你做了一个梦.....

更糊涂了.....

一脸懵逼

什么是递归

- 递归的组成要素
  - Base Case：问题最简单的情况，可以直接得到答案
    - 例子： Anyone born in the US is a natural born citizen
  - Recursive/Inductive case：使用一些简单的操作，将复杂的情况变成同一问题更简单的情况（simpler version of the same problem ）
    - 例子： anyone born outside the US……

中国人民大学
RENMIN UNIVERSITY OF CHINA

01. 初步递归

阶乘n!的递归求解

$$阶乘n!=n*(n-1)*...*1$$

- 你能以递归的方式定义问题吗?
  - Base case：n=1时，直接返回结果1
  - Inductive case: n>1时

$$n!=n*(n-1)!$$

汉诺塔问题 (1)

- 在世界中心贝拿勒斯（在印度北部）的圣庙里，一块黄铜板上插着三根宝石针。印度教的主神梵天在创造世界的时候，在其中一根针上从下到上地穿好了由大到小的64片金片，这就是所谓的汉诺塔。不论白天黑夜，总有一个僧侣在按照下面的法则移动这些金片：一次只移动一片，不管在哪根针上，小片必须在大片上面。僧侣们预言，当所有的金片都从梵天穿好的那根针上移到另外一根针上时，世界就将在一声霹雳中消灭，而梵塔、庙宇和众生也都将同归于尽。

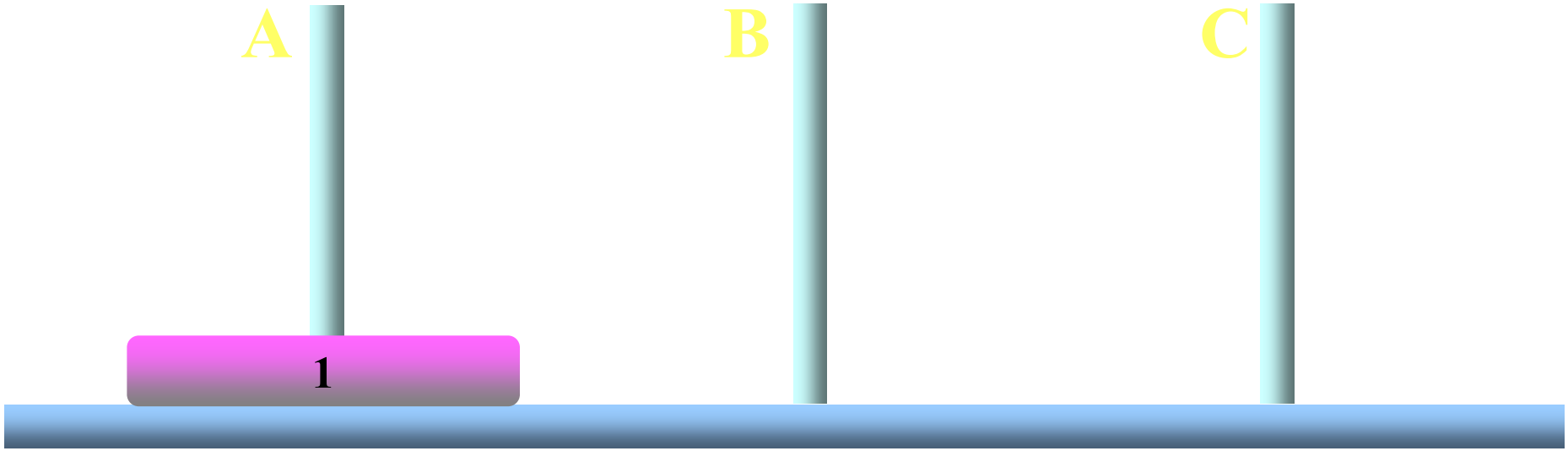
汉诺塔问题 (2)

¥9.80 包邮　　800+人付款

汉诺塔木制10层8层十层益智儿童汉罗塔玩具小学生逻辑思维训练

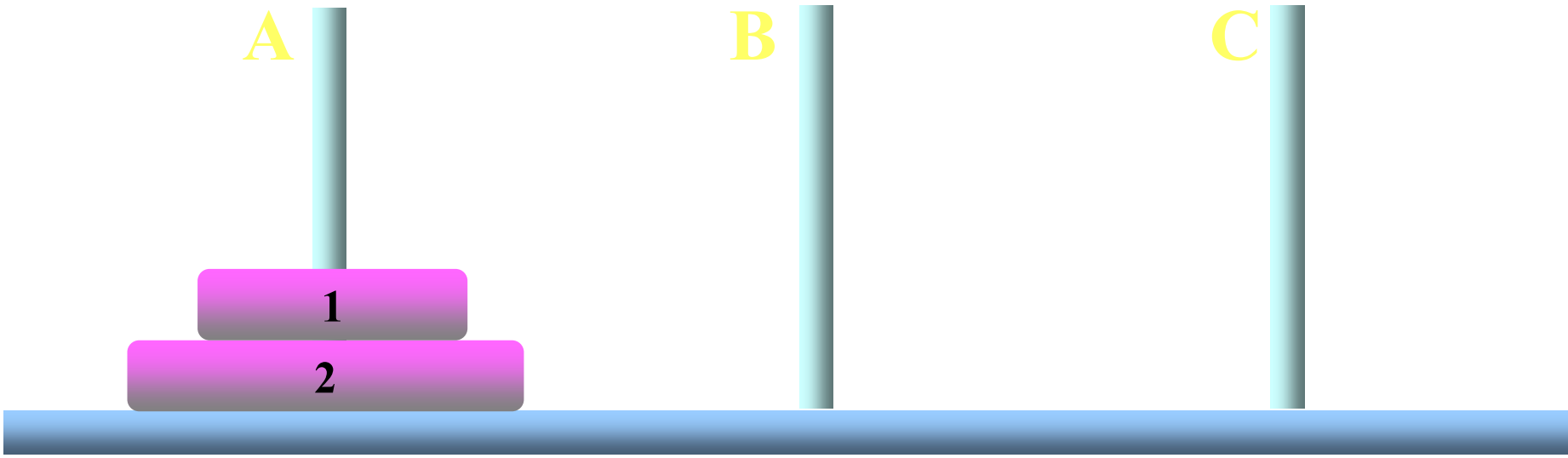
解题思路 (1)

- Base Case: 只有一个盘子的情况
  - 在A柱上只有一只盘子，假定盘号为 1，这时只需将该盘从 A 搬至 C，一次完成，记为 move 1 from A to C



解题思路 (2)

- 在 A 柱上有二只盘子，1 为小盘，2 为大盘
  - 将1号盘从A移至B，这是为了让 2号盘能移动 → **move 1 from A to B;**
  - 将 2 号盘从A 移至 C → **move 2 from A to C;**
  - 将 1 号盘从 B 移至 C → **move 1 form B to C;**



解题思路 (3)

- 假设在 A 柱上有三只盘子，1 为小盘，2 为中盘，3为大盘，怎么处理?

假设规模小的盘子移动问题

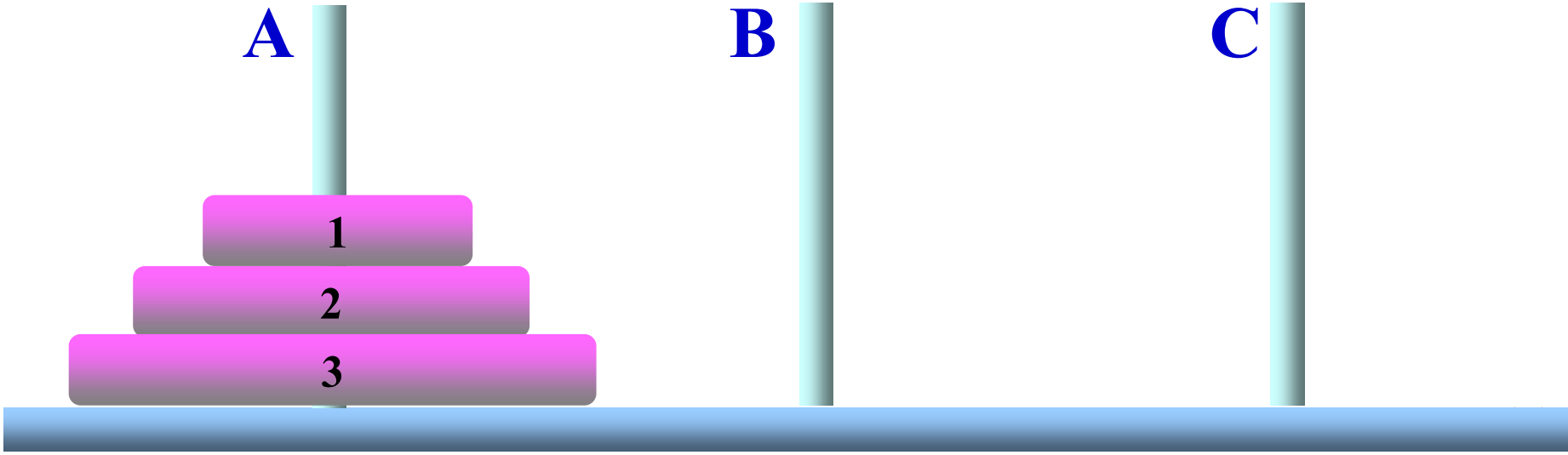
已经解决!

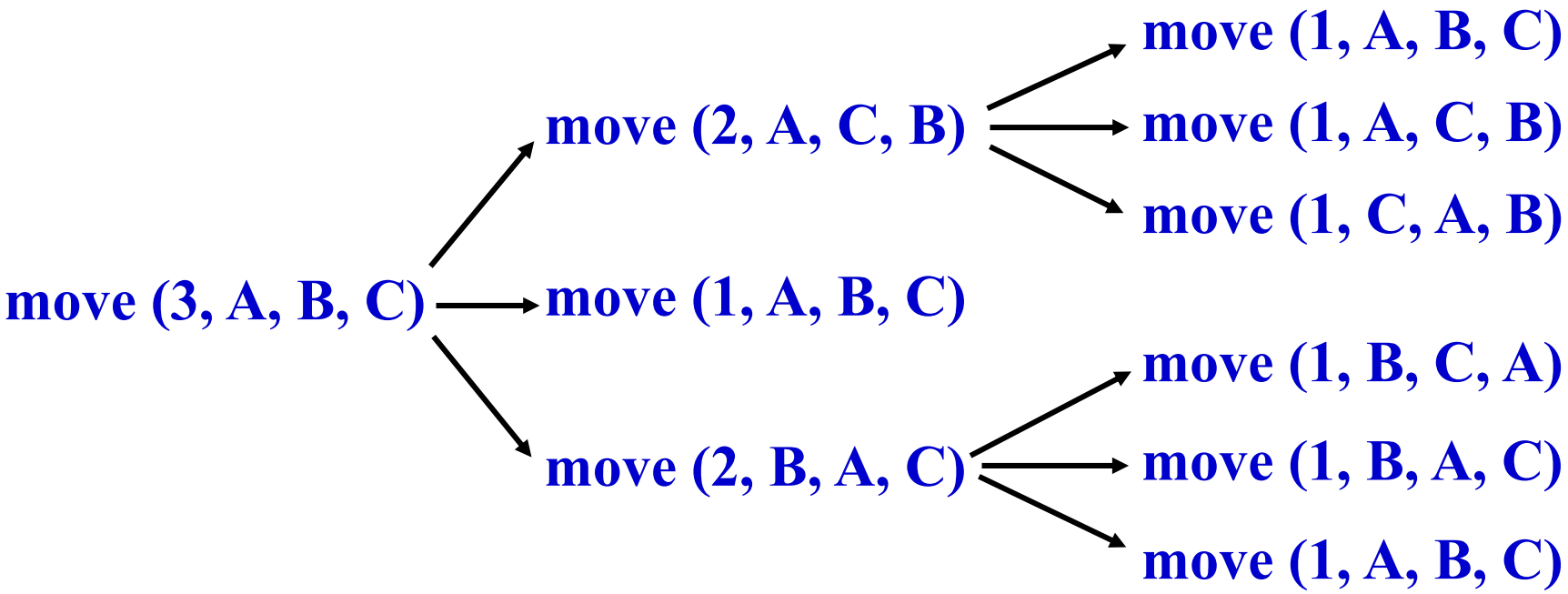
- 现将A柱上的1、2号盘子移到B柱上
- 把3号盘子移动到C柱上
- 再把1、2号盘子从B柱上移动到C柱上

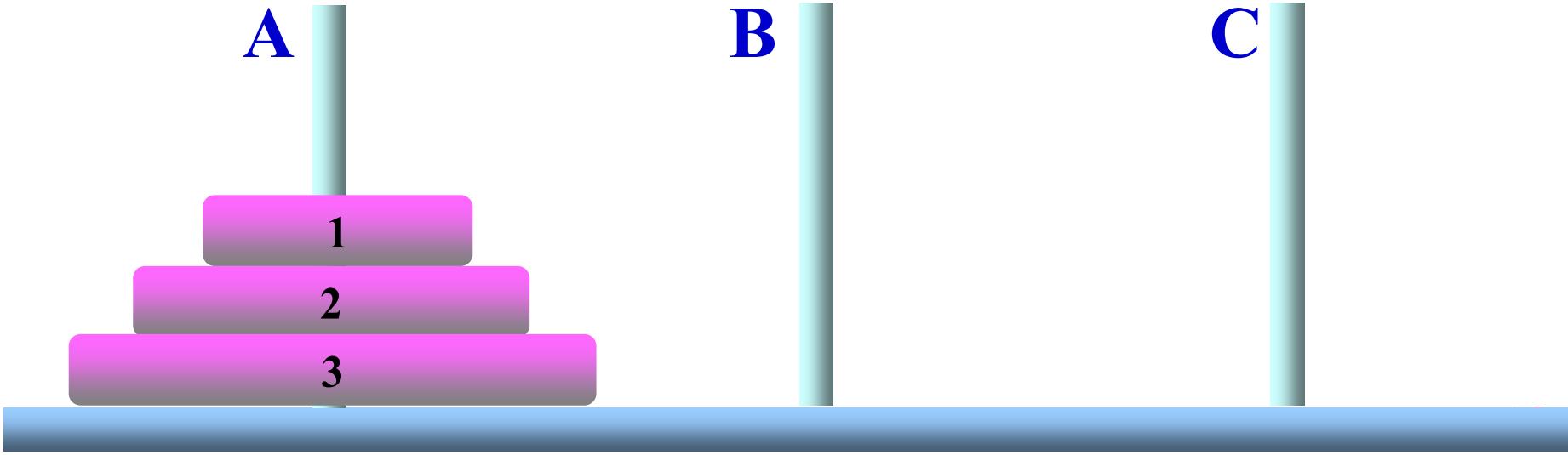
演示：移动3个盘子的分解

move (3, A, B, C)

- move (2, A, C, B)
- move (1, A, B, C)
- move (2, B, A, C)




move (3, A, B, C)
move (2, A, C, B)
move (1, A, B, C)
move (2, B, A, C)
move (1, A, B, C)
move (1, A, C, B)
move (1, C, A, B)
move (1, B, C, A)
move (1, B, A, C)
move (1, A, B, C)


A
B
C
1
2
3

判断回文

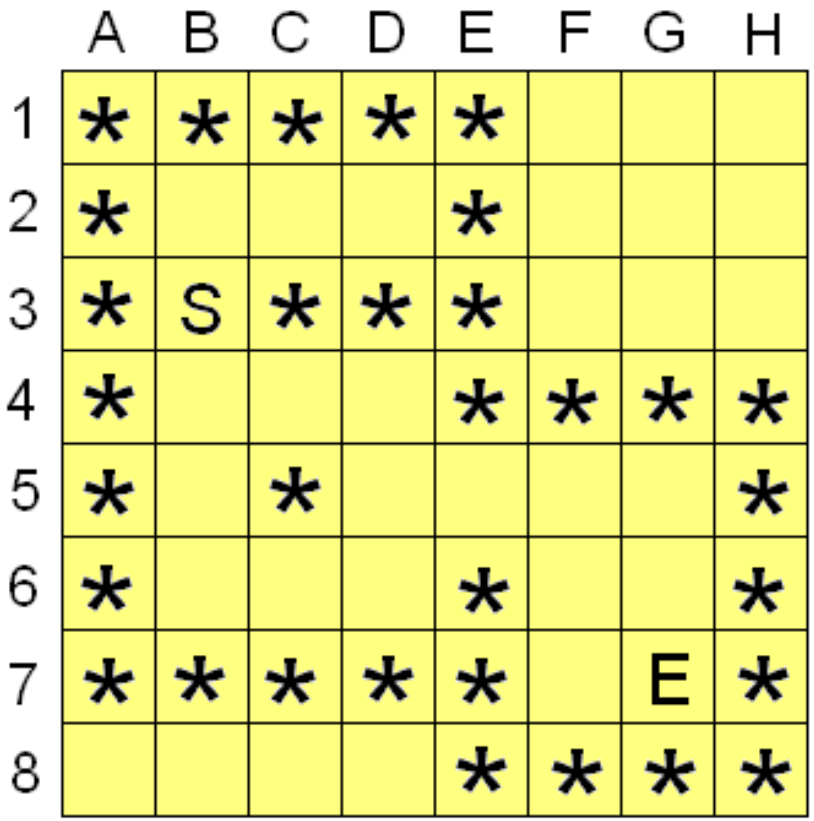
- 回文例子
  - Madam, I am Adam
  - Able was I ere I saw Elba
- 如何使用递归实现
  - **Base Case:** 空字符串/长度为1的字符串
  - **Inductive Case：**一个字符串是回文，当且仅当：
    - 该字符串第一个与最后一个字符相同
    - 中间的字符串是回文字符串

斐波那契数列递归解法

```
int FibRecur (int n) {
  if (n == 1 || n == 2) return 1;
  return (FibRecur(n-1) + FibRecur(n-2));
}
```

- Inductive Case
  - FibRecur(n-1) + FibRecur(n-2)
- Base Case
  - if (n == 1 || n == 2) return 1

迷宫(Maze)



从S点走到E点，不能经过任何的*符号的方格

迷宫(Maze)

- 做一个判定性问题：迷宫走得通吗?
  - isMazeSolveable
- 迷宫递归解法思路
  - **Base Cases:**
    - 走到了E点，返回1
    - 走到*点，返回0
  - **Inductive Cases**
    - 从当前点开始能走通，只要东南西北四个方向只好有一个能走通

略微扯远点

- 人生也是个多决策问题
- **Base Case**
  - 迎娶白富美，走向人生巅峰……
  - 遭遇失败与挫折
- **Inductive Case**
  - 当前的选择是否正确，取决于之后的选择是否正确……

中国人民大学
RENMIN UNIVERSITY OF CHINA

2. 枚举、递推

思考题

- 写一个计算阶乘的函数

$$f(n) = n!$$

- 下面三个公式思考方式有何不同?
  - 方式1： f(n) = 1 * … * (n-1) * n
  - 方式2： f(n) = (n-1)! * n
  - 方式3： f(n) = n * f(n-1)

阶乘n!的求解 – 枚举

```
int fact(int n)
{
    int m = 1;
    for (int i=1; i<=n; i++)
        m = m * i;
    return m;
}
```

- 根据公式，利用循环，进行枚举
  - 从数字1开始，一直枚举到数据n
  - 将枚举的数字乘起来得到结果

阶乘n!的求解 – 递推

```
int fact(int n) {
  int m[10];  // 假设n不超过10
  m[1] = 1;   // 递推的起始值
  for (int i=2; i<=n; i++)
      m[i] = m[i-1] * i;
  return m[n];                    /// 返回递推的终值
}
```

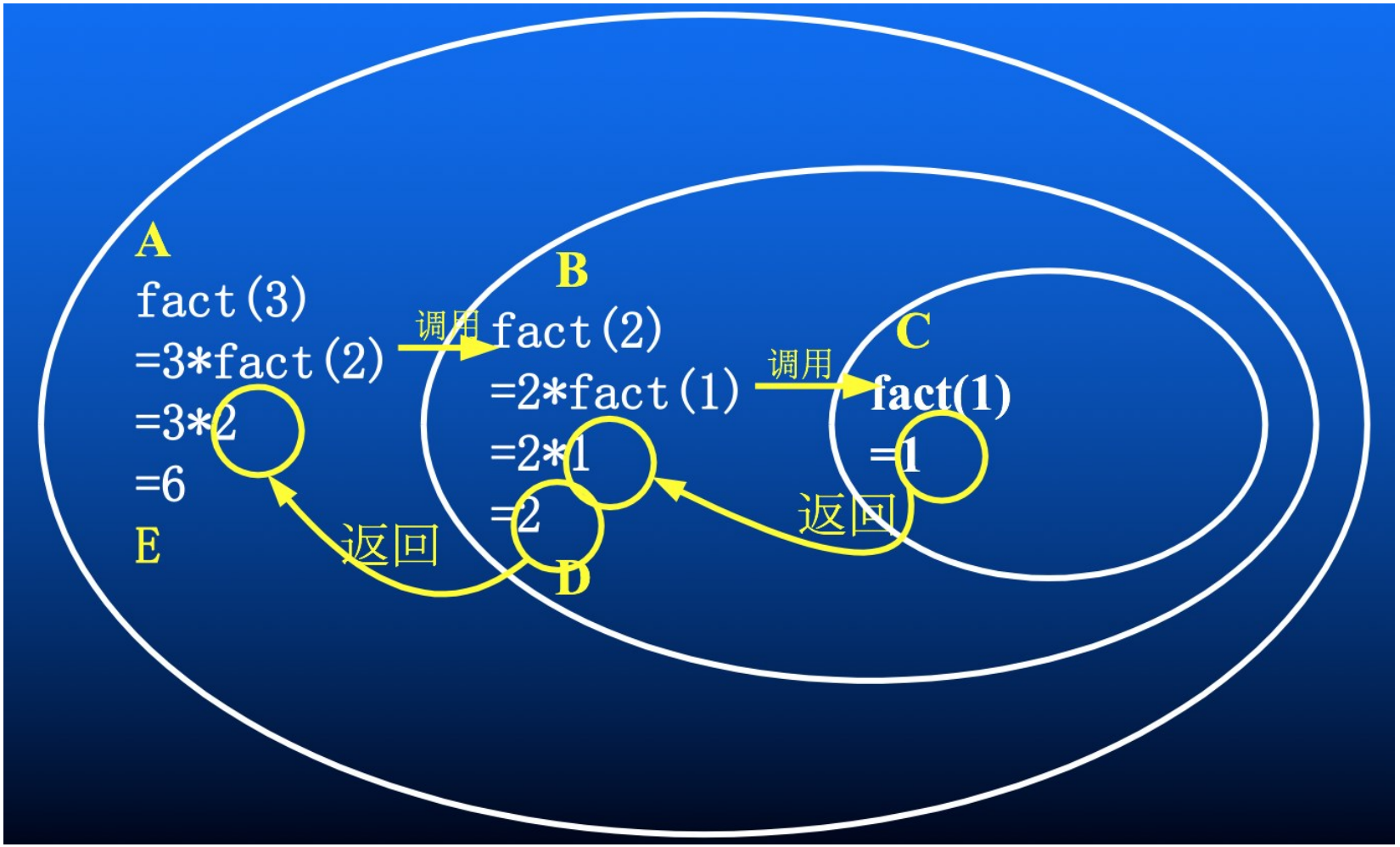
- 递推数列
  - 从某一项起，任何一项都可以用它前面的若干项来确定，这样的数列称为递推数列
  - 阶乘n!，可以再求得前一项(n-1)!后求得

阶乘n!的求解 – 递归

```
int fact(int n){
  if (n == 1)  return 1; // 递归的终止条件
  return n * fact(n-1);  // 直接返回
}  /// 自己调用自己：递归
```

- **递归：**
  - 核心：自己调用自己
  - 欲求阶乘n!，先调用 (n-1)!的结果

下面是fact(3)的调用和返回的示意图



从图可以想象：

欲求 fact(3)，先要求 fact(2)；要求 fact(2) 先求 fact(1)。

就象剥一颗圆白菜，

从外向里，一层层剥下来，到了菜心，遇到 1 的阶乘，其值为1，到达了递归的边界。

然后再用 fact(n)=n*fact(n-1) 这个普遍公式，从里向外倒推回去得到 fact(n) 的值。

这个图中“内层”与“外层”有着相同的结构。它们之间“你中有我，我中有你”，呈现相互依存的关系。

思考

- 写一个求解斐波那契数的函数?
  - 给定一个数字n，求第n位，斐波那契数列的值?
  - 问题1：用枚举可以求解吗?
  - 问题2：用递推可以求解吗?
  - 问题3：用递归可以求解吗?

斐波那契数列枚举解法

```
int FibEnum (int n) {
  if (n==1 || n == 2) return 1;
  int fn1 = 1, fn2 = 1;
  int fn;
  for (int i = 3; i <= n; i ++) {
      fn = fn1 + fn2;
      fn2 = fn1;
      fn1 = fn;
  }
  return fn;
}
```

- 利用循环枚举所有情况

斐波那契数列递推解法

```
int FibIter (int n) {
  int fib[MAX_N];
  for （int i = 0; i < n; i ++) {
    if (i == 0 || i == 1) fib[i] = 1;
    else fib[i] = fib[i-1] + fib[i-2];
  }
  return fib[n-1];
}
```

- 在计算第i项的数值之前，已经计算出i-1项和i-2项，并且存储在数组中。

斐波那契数列递归解法

```
int FibRecur (int n) {
  if (n == 1 || n == 2) return 1;
  return (FibRecur(n-1) + FibRecur(n-2));
}
```

- 把问题分解更小规模的问题
  - FibRecur(n-1) + FibRecur(n-2)
- 给出边界情况的解
  - if (n == 1 || n == 2) return 1

理解递归

- 递归并不符合我们的思维习惯
  - 日常生活中我们习惯一步接一步地做事
- 递归是一种基于函数调用的表达方式
  - 将解决某一问题抽象为函数
  - 把规模大的问题转化为规模小的子问题来解决
  - 函数自己调用自己
- 递归往往能以很简洁的代码，来解决非常复杂的问题

3. 函数变量

变量的四类存储属性

- 变量是对程序中数据的存储空间的抽象。数据类型是变量的操作属性。

存储属性	register	auto	static	extern
存储位置	寄存器	主存		
生存期	动态生存期		永久生存期	
作用域	局部		局部或全局	全局

动态变量

- 动态变量是在程序执行的某一时刻被动态地建立并在另一时刻被动态地撤销的一种变量。它们存在于程序的局部，也只在这个局部中可以使用。
- 动态变量有两种：
  - 自动变量（auto）
  - 寄存器变量（register）

自动变量（auto）

- 建立和撤销这些变量都是系统在程序执行过程中自动进行的，所以称之为自动变量。
- 自动变量的说明形式：

[auto]数据类型 变量名[=初值表达式]，…；

其中：auto为自动变量的存储类型标识符，如果省略auto系统隐含认为此变量是auto类型。

自动变量的说明:

1.自动变量是局部变量

main函数

```
x = 1
        x = 3
        call Prt
        printf( "2nd x= %d \n", x );
printf( "1st x = %d \n", x );
```

Prt函数

```
x = 5
printf( "3th x = %d \n", x );
```

自动变量的特点：

- C语言规定，如果内层与外层有相同名字的变量，则在内层范围内只有内层的变量有效，外层的同名变量在此范围内无效，或者说，外层的变量被内层的同名变量“屏蔽”掉了。
- 自动变量的优点：
  - 可节省存储空间；
  - 独立地在本区域命名变量；

自动变量的说明：（续2）

2.在对自动变量赋值前，它的值不确定。

- 例 使用未赋值的自动变量。

```
int main( )
{
    int i;
    printf( "i = %d \n",  i );
    return 0;
}
```

- 上述程序的运行结果不可预知。
- 注意：对于自动变量，必须对其赋初值后，才能引用它。

寄存器变量（register）

- 当把一个变量指定为寄存器存储类别时，系统就将它存放在CPU的一个寄存器中。
- 各CPU寄存器的个数和长度不同，因此，C标准对寄存器存储类别只给出建议，不作硬性规定。
- 在程序中如遇到指定为register类别的变量，系统会努力去实现它，但如果因条件所限不能实现时，系统会自动将它们处理成auto变量。

静态变量（static）

- static变量的存储空间在程序的整个运行期间是固定的、有效的，而不像动态变量是在程序执行中被动态建立和动态撤销的。
- static变量在编译时就为其分配存储空间，程序一开始执行便被建立，直到程序执行结束都是存在的。
- static变量的初始化是在编译时进行的。在定义时只能使用常量或常量表达式进行初始化。未显示初始化时，编译将把它们初始化为0（对int型）或0.0（对float型）。
- 在函数多次调用的过程中，静态局部变量的值具有继承性。但其值只能在本函（或分程序）中使用。

static变量的定义格式：

static 数据类型　变量名[=初始化常数表达式]，…；

```
void increment( void )
{
        static int x=0;
        x++;
        printf("%d\n",x);
}
int main( )
{
        increment();
        increment();
        increment();
        return 0;
}
```

- 运行结果:

  1

  2

  3

外部变量（extern）

- 定义在所有函数之外的变量称为外部变量。
- 外部变量是全局变量，它的作用域是从定义的位置开始到本文件的结束。在一个函数中改变外部变量的值，那么其后引用该变量时，得到的值是已被改变的值。
- 外部变量可以不出现在文件的起始部分，而出现在函数之间。在此之前的函数是不能引用该外部变量的。
- 外部变量的初始化，可以不在的定义处。

p *例8.21*

外部变量使用情况：

1.限定本文件的外部变量只在本文件中使用。

- 如果有的外部变量只允许本文件使用而不允许其它文件使用，则可以在此外部变量前加一个static，使用其局部化，称为静态外部变量。
- 内存的数据区分两个部分：
  - 静态存储区：外部变量、extern、static
  - 动态存储区：auto、形参

2.将外部变量的作用域在本文件范围内扩充。

- 对于位于定义点之前的函数，可以用extern说明符使变量的作用域扩充到需要用到它的函数。

外部变量使用情况：

1. 限定本文件的外部变量只在本文件中使用。
   - 如果有的外部变量只允许本文件使用而不允许其它文件使用，则可以在此外部变量前加一个static，使用其局部化，称为静态外部变量。
   - 内存的数据区分两个部分：
     - 静态存储区：外部变量、extern、static
     - 动态存储区：auto、形参

2.将外部变量的作用域在本文件范围内扩充。
   - 对于位于定义点之前的函数，可以用extern说明符使变量的作用域扩充到需要用到它的函数。

3.可以将外部变量的作用域扩充到其它文件。这时在需要用到这些外部变量的文件中用extern对变量进行声明即可。

外部变量的副作用

- 外部变量为公共信息的一种载体，虽然给程序设计带来一些方便，但也会产生一些副作用。在程序设计中应有限制地使用外部变量。
- 外部变量的副作用主要表现：
  - 占用固定的存储空间，降低了内存的利用率；
  - 增加函数（或程序）之间的联系，降低了模块的独立性。

存储类别小结

- 对一个数据的定义需要指定两种属性：
  - 数据类型：int、float、char、double、long、、short、unsigned
  - 存储类别：static、auto、register、extern
- 理解一个数据从两方面：
  - 作用域：全局/局部
  - 生存期：程序块/整个程序

作用域（空间）

局部变量
- auto变量，动态局部变量（离开函数，变量消失）
- static局部变量（离开函数，变量仍保留）
- register变量（离开函数，变量消失）
- 形式参数（或定义为auto或register变量）

全局变量
- 静态外部变量（只限本文件引用）
- 外部变量（即非静态的外变量,允许其它文件引用）

生存期（时间）

- 动态存储
  - auto变量（本函数内有效）
  - register变量（本函数内有效）
  - 形式参数
- 静态存储
  - 静态局部变量（函数内有效）
  - 静态外部变量（本文件内有效）
  - 外部变量（其它文件可以引用）

变量存储位置

内存中静态存储区
- 静态局部变量
- 静态外部变量（函数外部静态变量）
- 外部变量（可以为其它文件引用）

内存中动态存储区
- auto变量
- 形式参数

CPU中的寄存器：register变量

static的作用：

- static对局部变量和全局变量的作用不同。
  - 对于局部变量来说，它使变量由动态存储方式改变为静态存储方式。
  - 对全局变量来说，它使变量局部化（局部于本文件），但仍为静态存储方式。
- 从使用域看，凡有static说明的，其作用域是局限的，或者是局限于本函数内（静态局部变量），域者局限于本文件内（静态外部变量）。

extern作用:

- extern是变量的说明或声明符号，不是变量的定义符号。
- 它声明变量是在本函数或本文件以外定义的，本函数或本文件可以使用这些变量。

内部函数

- 如果一个函数只能被本文件中的函数调有，它称为内部函数。
- 在定义内部函数时，在函数名和函数类型的前面加static。

static 类型标识符 函数名（形参表）

- 例：static fun(int a, int b)

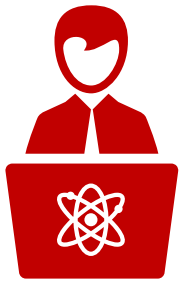
外部函数

- 在定义函数时，如果在函数首部的最左端冠以关键字extern，则表示此函数是外函数，可供其它文件调用。
- 如：

```
extern int func( int a, int b )
```

- C语言规定，如果在定义函数时省略extern，则隐含为外部函数。
- 在需要调用此函数的文件中，用extern声明所用的函数是外部函数。

中國人民大學
RENMIN UNIVERSITY OF CHINA



谢谢大家！