第7章 用函数实现模块化程序设计

7.1为什么要用函数

- 问题：
  - 如果程序的功能比较多，规模比较大，把所有代码都写在**main**函数中，就会使主函数变得庞杂、头绪不清，阅读和维护变得困难
  - 有时程序中要多次实现某一功能，就需要多次重复编写实现此功能的程序代码，这使程序冗长，不精炼

7.1为什么要用函数

- 解决的方法：用模块化程序设计的思路
  - 采用“组装”的办法简化程序设计的过程
  - 事先编好一批实现各种不同功能的函数
  - 把它们保存在函数库中，需要时直接用

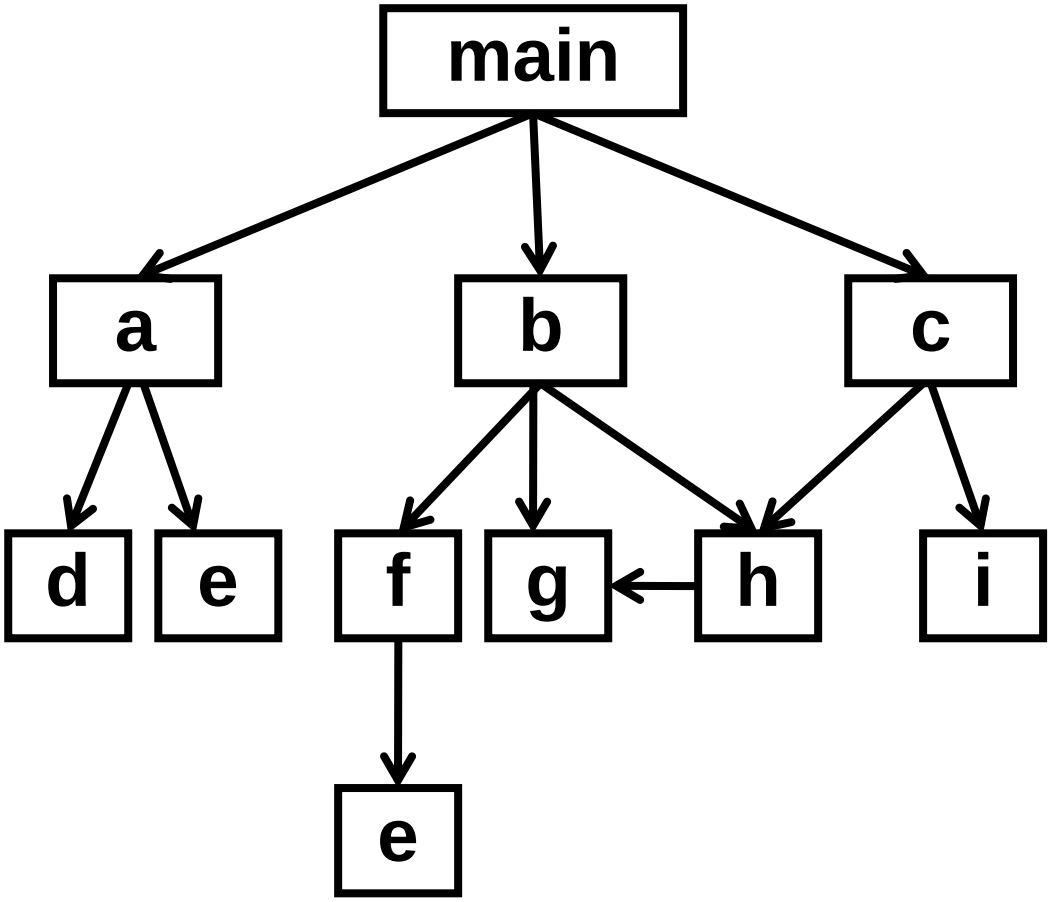
7.1为什么要用函数

- 解决的方法：用模块化程序设计的思路
  - 函数就是功能
  - 每一个函数用来实现一个特定的功能
  - 函数的名字应反映其代表的功能

7.1为什么要用函数

- 在设计一个较大的程序时，往往把它分为若干个程序模块，每一个模块包括一个或多个函数，每个函数实现一个特定的功能
- C程序可由一个主函数和若干个其他函数构成
- 主函数调用其他函数，其他函数也可以互相调用
- 同一个函数可以被一个或多个函数调用任意多次

7.1为什么要用函数



7.1为什么要用函数

- 可以使用库函数
- 可以使用自己编写的函数
- 在程序设计中要善于利用函数，可以减少重复编写程序段的工作量，同时可以方便地实现模块化的程序设计

7.1为什么要用函数

例7.1 输出以下的结果，用函数调用实现。

****

How do you do!

****

7.1为什么要用函数

- 解题思路：
  - 在输出的文字上下分别有一行“*”号，显然不必重复写这段代码，用一个函数**print_star**来实现输出一行“*”号的功能。
  - 再写一个**print_message**函数来输出中间一行文字信息
  - 用主函数分别调用这两个函数

```
#include <stdio.h>
void print_star();
void print_message();
int main()
{print_star();
   print_message();
   print_star();
   return 0;
}
void print_star()
{  printf("**********\n"); }
void print_message()
{  printf("  How do you do!\n");  }
```

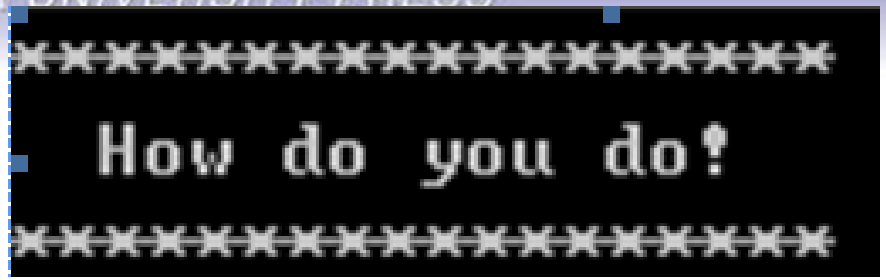
输出16个*

输出一行文字

```
#include <stdio.h>
int main()
{  void print_star();
   void print_message();
   print_star();   print_message();
   print_star();
   return 0;
}
void print_star()
{  printf("**********\n"); }
void print_message()
{  printf("  How do you do!\n");  }
```

声明函数

定义函数




```
#include <stdio.h>
int main()
{ void print_star();
  void print_message();
  print_star();  print_message();
  print_star();
  return 0;
}
void print_star()
{  printf("**********\n"); }
void print_message()
{  printf("  How do you do!\n");  }
```

- 说明：

(1) 一个C程序由一个或多个程序模块组成，每一个程序模块作为一个源程序文件。对较大的程序，一般不希望把所有内容全放在一个文件中，而是将它们分别放在若干个源文件中，由若干个源程序文件组成一个**C**程序。这样便于分别编写、分别编译，提高调试效率。一个源程序文件可以为多个**C**程序共用。

- 说明：

  (2) 一个源程序文件由一个或多个函数以及其他有关内容（如预处理指令、数据声明与定义等）组成。一个源程序文件是一个编译单位，在程序编译时是以源程序文件为单位进行编译的，而不是以函数为单位进行编译的。

- 说明：

  (3) C程序的执行是从**main**函数开始的，如果在**main**函数中调用其他函数，在调用后流程返回到**main**函数，在**main**函数中结束整个程序的运行。

- 说明：

(4) 所有函数都是平行的，即在定义函数时是分别进行的，是互相独立的。一个函数并不从属于另一个函数，即函数不能嵌套定义。函数间可以互相调用，但不能调用**main**函数。**main**函数是被操作系统调用的。

➢说明：

(5) 从用户使用的角度看，函数有两种。

◆库函数，它是由系统提供的，用户不必自己定义而直接使用它们。应该说明，不同的**C**语言编译系统提供的库函数的数量和功能会有一些不同，当然许多基本的函数是共同的。

◆用户自己定义的函数。它是用以解决用户专门需要的函数。

- 说明：

(6) 从函数的形式看，函数分两类。

① 无参函数。无参函数一般用来执行指定的一组操作。无参函数可以带回或不带回函数值，但一般以不带回函数值的居多。

② 有参函数。在调用函数时，主调函数在调用被调用函数时，通过参数向被调用函数传递数据，一般情况下，执行被调用函数时会得到一个函数值，供主调函数使用。

7.2 怎样定义函数

7.2.1 为什么要定义函数

7.2.2 定义函数的方法

7.2.1 为什么要定义函数

- C语言要求，在程序中用到的所有函数，必须“先定义，后使用”
- 指定函数名字、函数返回值类型、函数实现的功能以及参数的个数与类型，将这些信息通知编译系统。

7.2.1 为什么要定义函数

- 指定函数的名字，以便以后按名调用
- 指定函数类型，即函数返回值的类型
- 指定函数参数的名字和类型，以便在调用函数时向它们传递数据
- 指定函数的功能。这是最重要的，这是在函数体中解决的

7.2.1 为什么要定义函数

- 对于库函数，程序设计者只需用**#include**指令把有关的头文件包含到本文件模块中即可
- 程序设计者需要在程序中自己定义想用的而库函数并没有提供的函数

7.2.2 定义函数的方法

1.定义无参函数

定义无参函数的一般形式为:

```
类型名   函数名()
 {
      函数体
 }
```

```
类型名   函数名(void)
 {
      函数体
 }
```

包括声明部分和语句部分

7.2.2 定义函数的方法

1.定义无

指定函数值的类型

定义无参函数的一般形式为:

```
类型名   函数名()
 {
      函数体
 }
```

```
类型名   函数名(void)
 {
      函数体
 }
```

7.2.2 定义函数的方法

2.定义有参函数

定义有参函数的一般形式为:

```
类型名 函数名（形式参数表列）
{
    函数体
}
```

7.2.2 定义函数的方法

3. 定义空函数

定义空函数的一般形式为:

类型名 函数名（ ）

{ }

- 先用空函数占一个位置，以后逐步扩充
- 好处：程序结构清楚，可读性好，以后扩充新功能方便，对程序结构影响不大

7.3 调用函数

7.3.1函数调用的形式

7.3.2函数调用时的数据传递

7.3.3函数调用的过程

7.3.4函数的返回值

7.3.1函数调用的形式

- 函数调用的一般形式为：

  函数名（实参表列）

- 如果是调用无参函数，则“实参表列”可以没有，但括号不能省略
- 如果实参表列包含多个实参，则各参数间用逗号隔开

7.3.1函数调用的形式

- 按函数调用在程序中出现的形式和位置来分，可以有以下3种函数调用方式：

1．函数调用语句

- 把函数调用单独作为一个语句

  如printf_star();

- 这时不要求函数带回值，只要求函数完成一定的操作

7.3.1函数调用的形式

- 按函数调用在程序中出现的形式和位置来分，可以有以下3种函数调用方式：

2．函数表达式

- 函数调用出现在另一个表达式中

  如c=max(a,b);

- 这时要求函数带回一个确定的值以参加表达式的运算

7.3.1函数调用的形式

- 按函数调用在程序中出现的形式和位置来分，可以有以下3种函数调用方式：

3．函数参数

- 函数调用作为另一函数调用时的实参

  如m＝max(a,max(b,c));

- 其中max(b,c)是一次函数调用，它的值作为max另一次调用的实参

7.3.2 函数调用时的数据传递

1.形式参数和实际参数

- ◆在调用有参函数时，主调函数和被调用函数之间有数据传递关系
- ◆定义函数时函数名后面的变量名称为“形式参数”（简称“形参”）
- ◆主调函数中调用一个函数时，函数名后面参数称为“实际参数”（简称“实参”）
- ◆ 实际参数可以是常量、变量或表达式

7.3.2 函数调用时的数据传递

2. 实参和形参间的数据传递

- ◆在调用函数过程中，系统会把实参的值传递给被调用函数的形参
- ◆或者说，形参从实参得到一个值
- ◆该值在函数调用期间有效，可以参加被调函数中的运算

7.3.2 函数调用时的数据传递

例7.2 输入两个整数，要求输出其中值较大者。要求用函数来找到大数。

- 解题思路：

(1)函数名应是见名知意，今定名为max

(2) 由于给定的两个数是整数，返回主调函数的值（即较大数）应该是整型

(3)max函数应当有两个参数，以便从主函数接收两个整数，因此参数的类型应当是整型

7.3.2 函数调用时的数据传递

先编写max函数：

```
int max(int x,int y)
{
    // int z;
    // z=x>y?x:y;
     return(x>y?x:y);
}
```

7.3.2 函数调用时的数据传递

在max函数上面，再编写主函数

```
#include <stdio.h>
int main()
{ int max(int x,int y);   int a,b,c;
   printf("two integer numbers: ");
   scanf("%d,%d",&a,&b);
   c=max(a,b);
   printf("max is %d\n",c);
}
```

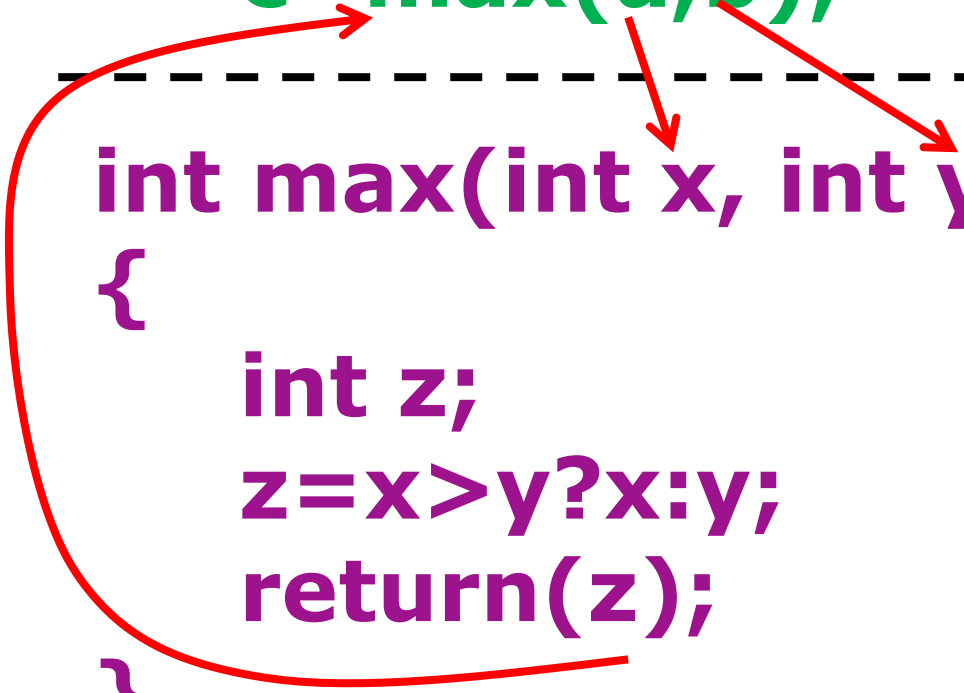
实参可以是常量、变量或表达式

```
two integer numbers:12,-34
max is 12
```

7.3.2 函数调用时的数据传递

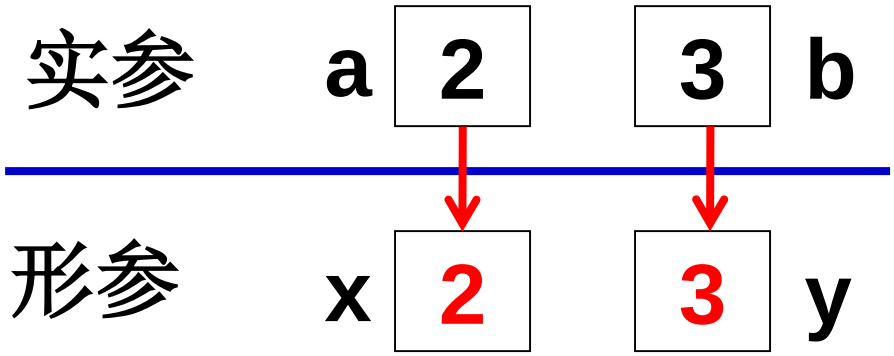
```
c=max(a,b);          （main函数）
```

```
int max(int x, int y)   （max函数）
{
    int z;
    z=x>y?x:y;
    return(z);
}
```



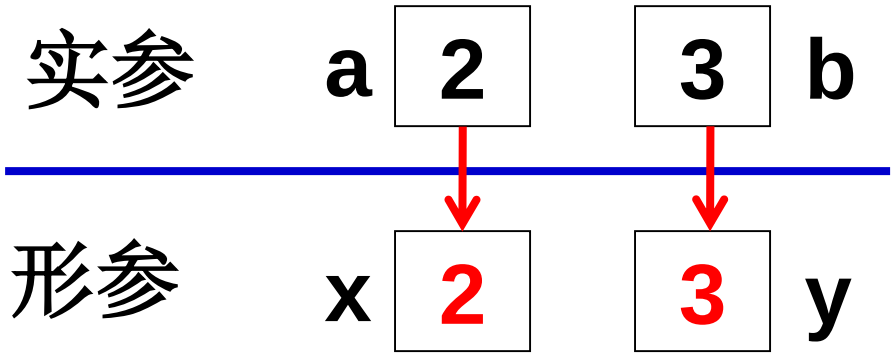
7.3.3 函数调用的过程

在定义函数中指定的形参，在未出现函数调用时，它们并不占内存中的存储单元。在发生函数调用时，函数max的形参被临时分配内存单元。



7.3.3 函数调用的过程

- 调用结束，形参单元被释放
- 实参单元仍保留并维持原值，没有改变
- 如果在执行一个被调用函数时，形参的值发生改变，不会改变主调函数的实参的值



7.3.4. 函数的返回值

➢通常，希望通过函数调用使主调函数能得到一个确定的值，这就是函数值(函数的返回值)

(1)函数的返回值是通过函数中的return语句获得的。

◆一个函数中可以有一个以上的return语句，执行到哪一个return语句，哪一个就起作用

◆return语句后面的括号可以不要

7.3.4. 函数的返回值

➢通常，希望通过函数调用使主调函数能得到一个确定的值，这就是函数值(函数的返回值)

(2) 函数值的类型。应当在定义函数时指定函数值的类型

7.3.4. 函数的返回值

➢通常，希望通过函数调用使主调函数能得到一个确定的值，这就是函数值(函数的返回值)

(3)在定义函数时指定的函数类型一般应该和return语句中的表达式类型一致

◆如果函数值的类型和return语句中表达式的值不一致，则以函数类型为准

7.3.4. 函数的返回值

例7.3将例7.2稍作改动，将在max函数中定义的变量z改为float型。函数返回值的类型与指定的函数类型不同，分析其处理方法。

➢解题思路：如果函数返回值的类型与指定的函数类型不同，按照赋值规则处理。

```
#include <stdio.h>
int main()
 { int max(float x,float y);
   float a,b;  int c;
   scanf("%f,%f,",&a,&b);
   c=max(a,b);
   printf("max is %d\n",c);
   return 0;
 }
int max(float x,float y)
 {  float z;
    z=x>y?x:y;
    return( z ) ;
}
```

```
1.5,2.6
max is 2
```

变为2

1.5 2.6

2.6 ⇒ 2

7.4对被调用函数的声明和函数原型

- 在一个函数中调用另一个函数需要具备如下条件：

  (1) 被调用函数必须是已经定义的函数（是库函数或用户自己定义的函数）

  (2) 如果使用库函数，应该在本文件开头加相应的**#include**指令

  (3) 如果使用自己定义的函数，而该函数的位置在调用它的函数后面，应该声明

7.4对被调用函数的声明和函数原型

例7.4 输入两个实数，用一个函数求出它们之和。

- 解题思路：用add函数实现。首先要定义add函数，它为float型，它应有两个参数，也应为float型。特别要注意的是：要对add函数进行声明。

7.4对被调用函数的声明和函数原型

- 分别编写**add**函数和**main**函数，它们组成一个源程序文件
- **main**函数的位置在**add**函数之前
- 在**main**函数中对**add**函数进行声明

```
#include <stdio.h>
int main()
{ float add(float x, float y);
  float a,b,c;
  printf("Please enter a and b:");
  scanf("%f,%f",&a,&b);
  c=add(a,b);
  printf("sum is %f\n",c);
  return 0;
}
```

对add函数声明

```
float add(float x,float y)
{ float z;
  z=x+y;
  return(z);
}
```

求两个实数之和，
函数值也是实型

```
#include <stdio.h>
int main()
{ float add(float x, float y);
  float a,b,c;
  printf("Please enter a and b:");
  scanf("%f,%f",&a,&b);
  c=add(a,b);
  printf("sum is %f\n",c);
  return 0;
}
```

只差一个分号

```
float add(float x,float y)
{ float z;
  z=x+y;
  return(z);
}
```

```
Please enter a and b:3.6,6.5
sum is 10.100000
```

```
#include
int main()
{ float add(float x, float y);
  float a,b,c;
  printf("Please enter a and b:");
  scanf("%f,%f",&a,&b);
  c=add(a,b);
  printf("sum is %f\n",c);
  return 0;
}
```

调用add函数

```
float add(float x,float y)
{ float z;
  z=x+y;
  return(z);
}
```

定义add函数

- 函数原型的一般形式有两种：

  如 **float add(float x, float y);**

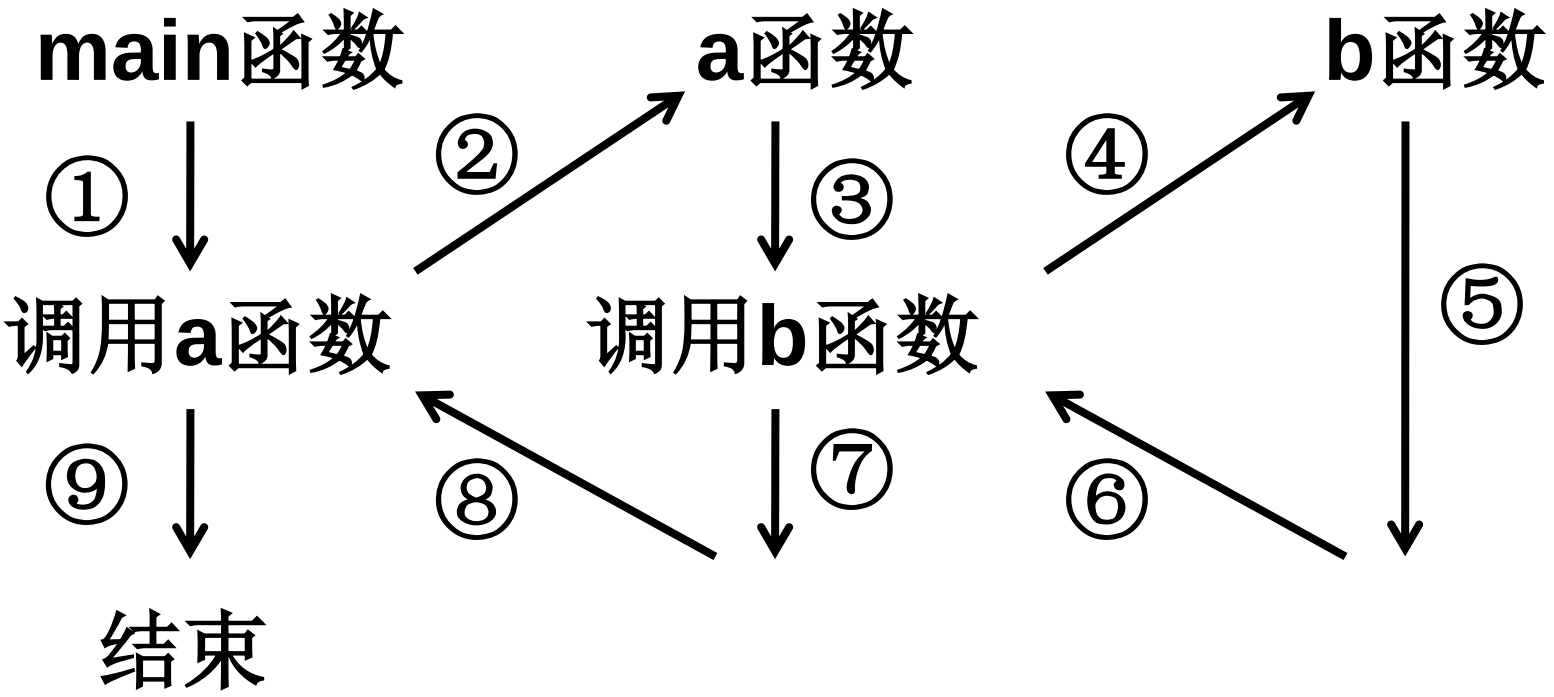
  float add(float, float);

- 原型说明可以放在文件的开头，这时所有函数都可以使用此函数

7.5 函数的嵌套调用

- C语言的函数定义是互相平行、独立的
- 即函数不能嵌套定义
- 但可以嵌套调用函数
- 即调用一个函数的过程中，又可以调用另一个函数

7.5 函数的嵌套调用



7.5 函数的嵌套调用

例7.5 输入4个整数，找出其中最大的数。用函数的嵌套调用来处理。

- 解题思路：
  - main中调用max4函数，找4个数中最大者
  - max4中再调用max2，找两个数中的大者
  - max4中多次调用max2，可找4个数中的大者，然后把它作为函数值返回main函数
  - main函数中输出结果

主函数

对max4 函数声明

```
#include <stdio.h>
int main()
{ int max4(int a,int b,int c,int d);
   int a,b,c,d,max;
   printf("4 interger numbers:");
   scanf("%d%d%d%d",&a,&b,&c,&d);
   max=max4(a,b,c,d);
   printf("max=%d \n",max);
   return 0;
}
```

主函数

```
#include <stdio.h>
int main()
{ int max4(int a,int b,int c,int d);
   int a,b,c,d,max;
   printf("4 interger numbers:");
   scanf("%d%d%d%d",&a,&b,&c,&d);
   max=max4(a,b,c,d);
   printf("max=%d \n",max);
   return 0;
}
```

输入4个整数

主函数

```
#include <stdio.h>
int main()
{ int max4(int a,int b,int c,int d);
   int a,b,c,d
   printf("4                bers:");
   scanf("%d%d%d%d",&a,&b,&c,&d);
   max=max4(a,b,c,d);
   printf("max=%d \n",max);
   return 0;
}
```

调用后肯定是4个数中最大者

输出最大者

max4函数

```
int max4(int a,int b,int c,int d)
{  int max2(int a,int b);
   int m;
   m=max2(a,b);
   m=max2(m,c);
   m=max2(m,d);
   return(m);
}
```

对max2 函数声明

max4函数

```
int max4(int a,int b,int c,int d)
{  int max2(int a,int b);
   int m;
   m=max2(a,b);
   m=max2(m,c);
   m=max2(m,d);
   return(m);
}
```

a,b中较大者

a,b,c中较大者

a,b,c,d中最大者

max4函数

```
int max4(int a,int b,int c,int d)
{  int max2(int a,int b);
   int m;
   m=max2(a,b);
   m=max2(m,c);
   m=max2(m,d);
   return(m);
}
```

找a,b中较大者

max2函数

```
int max2(int a,int b)
{  if(a>=b)
      return a;
   else
      return b;
}
```

max4函数

```
int max4(int a,int b,int c,int d)
{  int max2(int a,int b);
   int m;
   m=max2(a,b);
   m=max2(m,c);
   m=max2(m,d);
   return(m);
}
```

return(a>b?a:b);

max2函数

```
int max2(int a,int b)
{  if(a>=b)
      return a;
   else
      return b;
}
```

max4函数

```
int max4(int a,int b,int c,int d)
{  int max2(int a,int b);
   int m;
   m=max2(a,b);
   m=max2(m,c);
   m=max2(m,d);
   return(m);
}
```

```
int max2(int a,int b)
{    return(a>b?a:b);   }
```

max4函数

```
int max4(int a,int b,int c,int d)
{  int max2(int a,int b);
   int m;
   m=max2(a,b);
   m=max2(m,c);
   m=max2(m,d);
   return(m);
}
```

m=max2(max2(a,b),c);

```
int max2(int a,int b)
{    return(a>b?a:b);   }
```

max4函数

```
int max4(int a,int b,int c,int d)
{  int max2(
   int m;
   m=max2(a,b);
   m=max2(m,c);
   m=max2(m,d);
   return(m);
}
```

m=max2(max2(max2(a,b),c),d);

```
int max2(int a,int b)
{   return(a>b?a:b);  }
```

max4函数

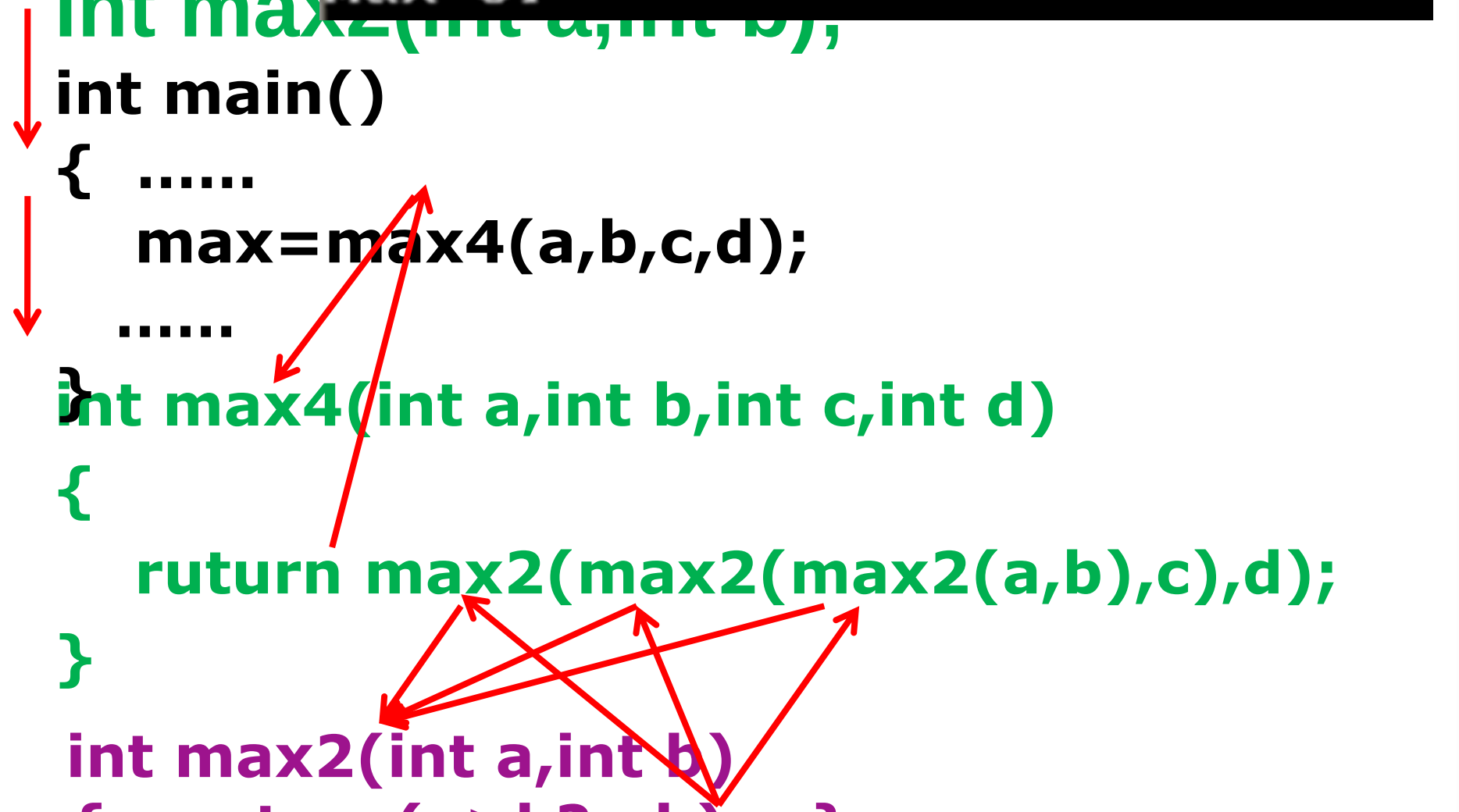
```
int max
{  int max2(i      ,int b);
   int m;
   m=max2(a,b);
   m=max2(m,c);
   m=max2(m,d);
   return(m);
}
```

ruturn max2(max2(max2(a,b),c),d);

```
int max2(int a,int b)
{    return(a>b?a:b);   }
```

```
4 interger numbers:12 45 -6 89
max=89
```

```
#include
int max2(int a,int b);
int main()
{  ……
   max=max4(a,b,c,d);
  ……
}
int max4(int a,int b,int c,int d)
{
   ruturn max2(max2(max2(a,b),c),d);
}
int max2(int a,int b)
{   return(a>b?a:b);  }
```



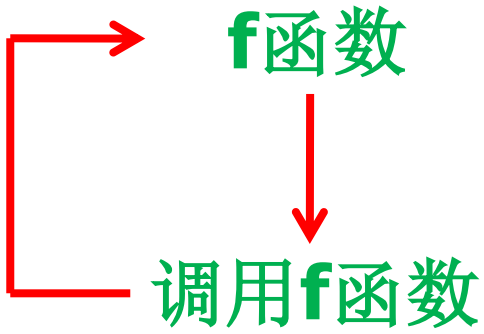
7.6 函数的递归调用

- 在调用一个函数的过程中又出现直接或间接地调用该函数本身，称为函数的递归调用。
- C语言的特点之一就在于允许函数的递归调用。

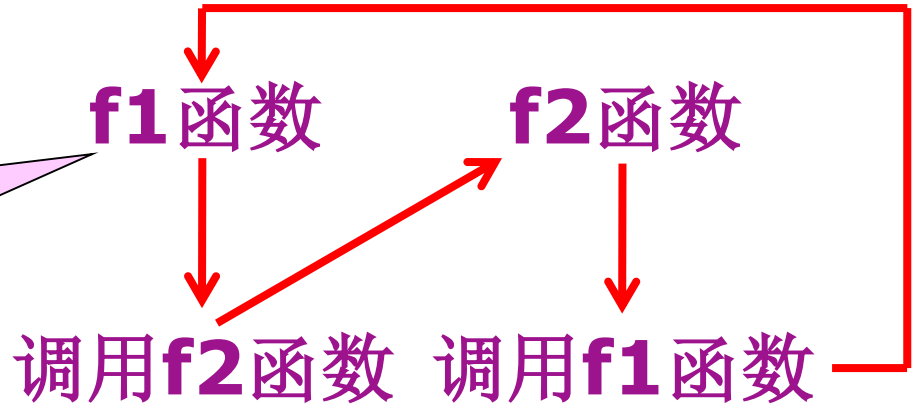
7.6 函数的递归调用

```
int f(int x)
{
    int y,z;
    z=f(y);
    return (2*z);
}
```

直接调用本函数



间接调用本函数



应使用if语句控制结束调用

7.6 函数的递归调用

例7.6 有5个学生坐在一起

- 问第5个学生多少岁？他说比第4个学生大2岁
- 问第4个学生岁数，他说比第3个学生大2岁
- 问第3个学生，又说比第2个学生大2岁
- 问第2个学生，说比第1个学生大2岁
- 最后问第1个学生，他说是10岁
- 请问第5个学生多大

7.6 函数的递归调用

- 解题思路：
  - 要求第 5 个年龄，就必须先知道第 4 个年龄
  - 要求第 4 个年龄必须先知道第 3 个年龄
  - 第 3 个年龄又取决于第 2 个年龄
  - 第 2 个年龄取决于第 1 个年龄
  - 每个学生年龄都比其前 1 个学生的年龄大 2

7.6 函数的递归调用

➢解题思路：

age(5)=age(4)+2

age(4)=age(3)+2

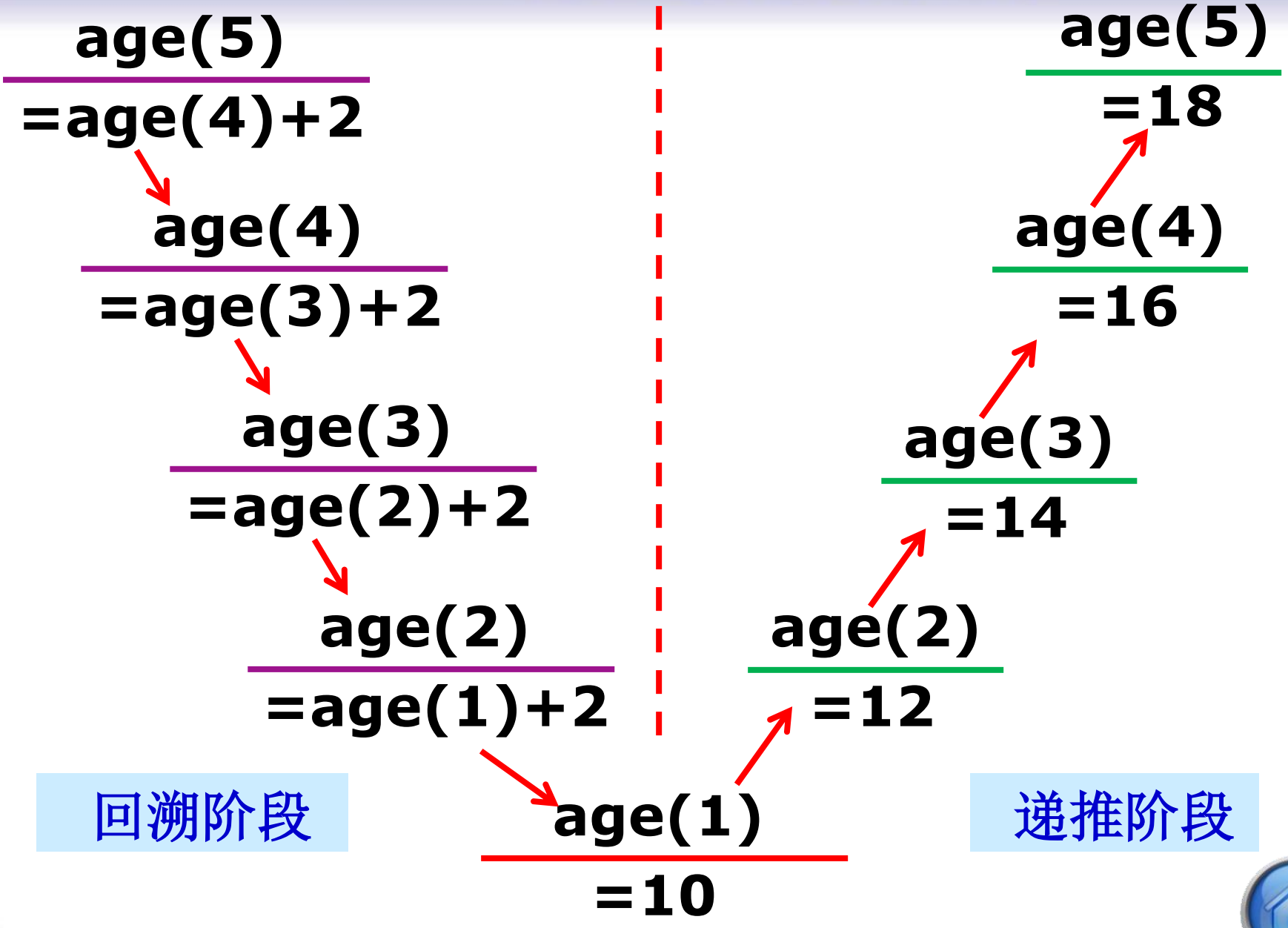
age(3)=age(2)+2

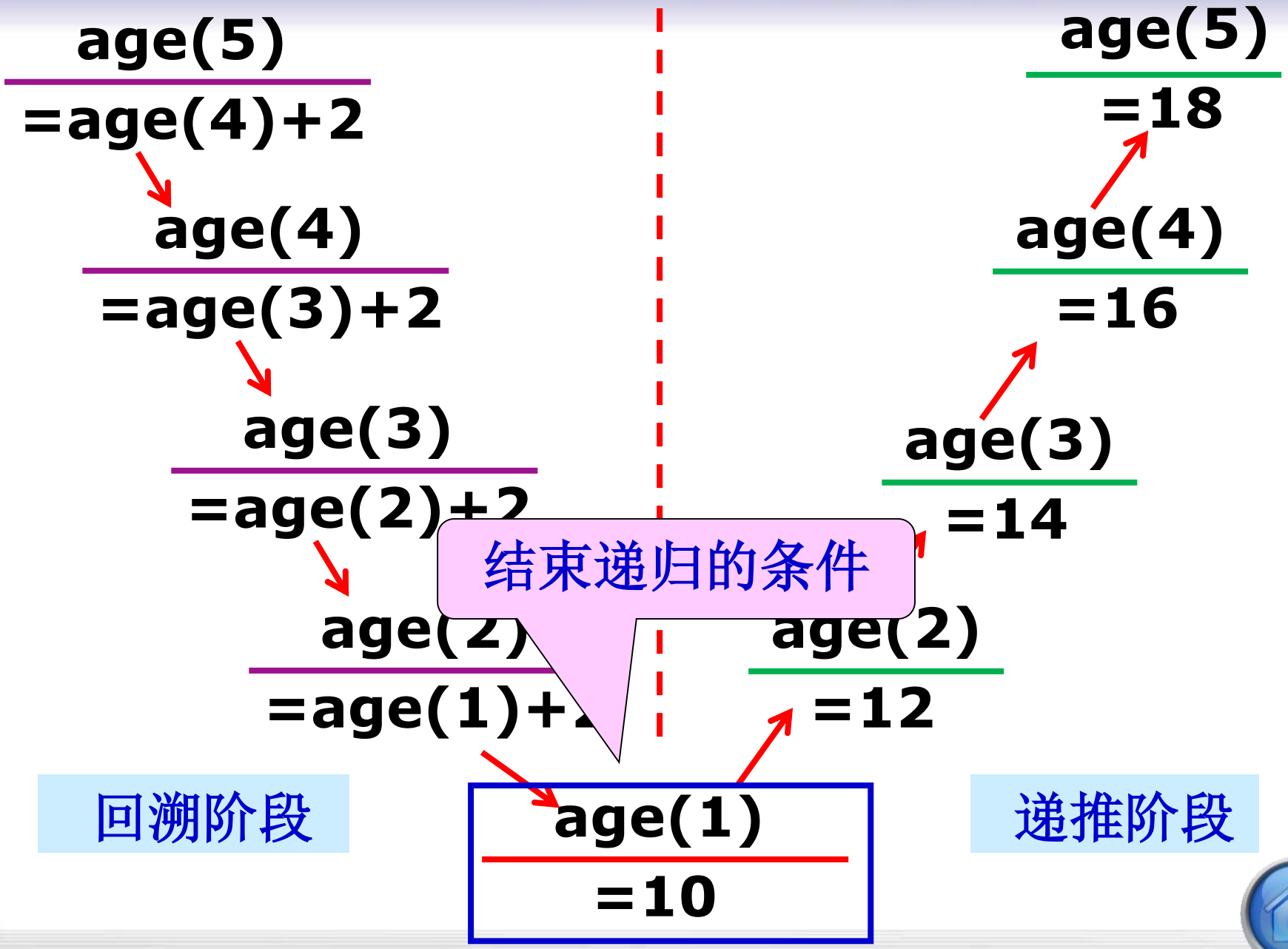
age(2)=age(1)+2

age(1)=10

$$age(n)=10 \qquad (n=1)$$

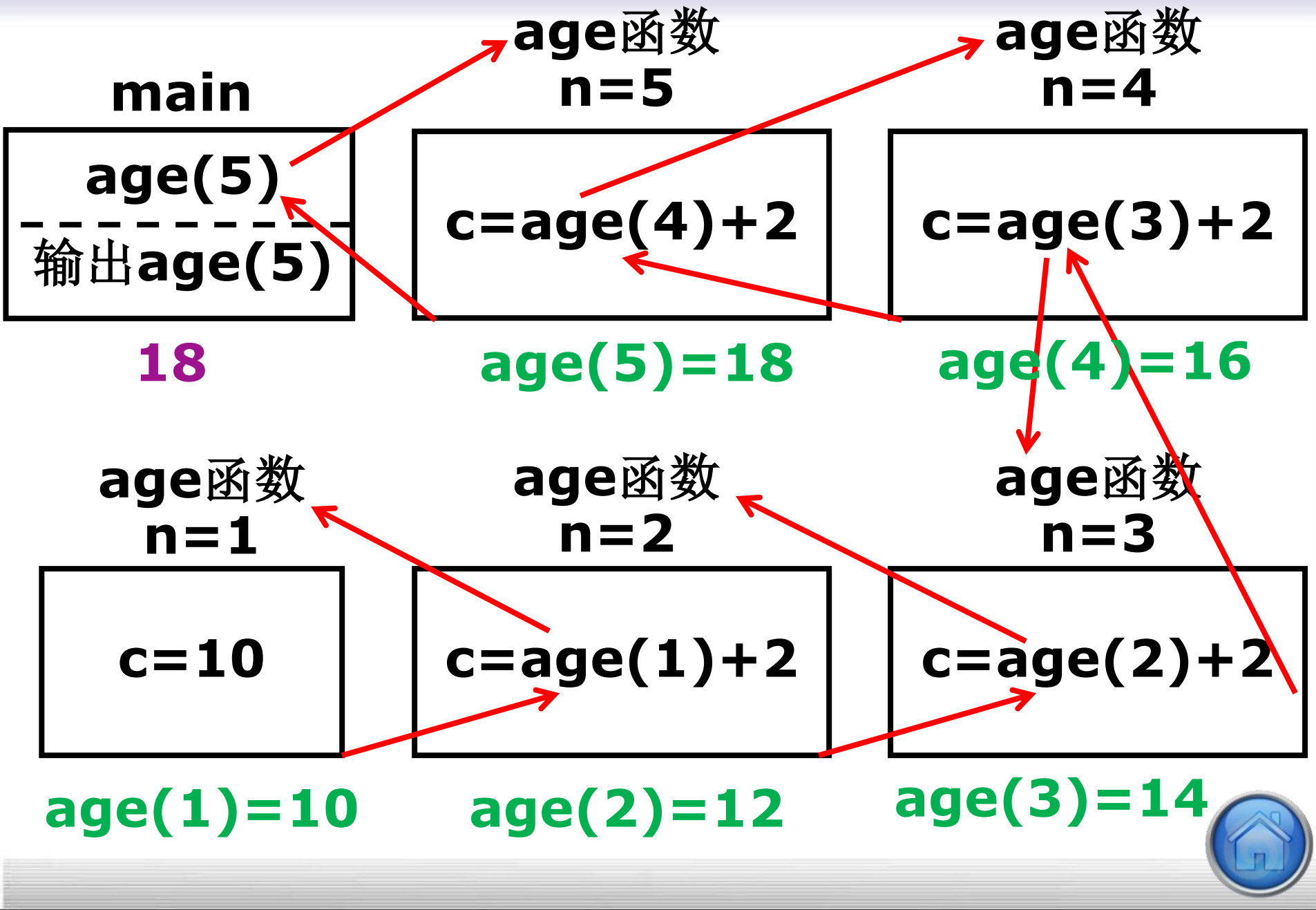
$$age(n)=age(n-1)+2 \qquad (n>1)$$


age(5)
=age(4)+2
age(4)
=age(3)+2
age(3)
=age(2)+2
age(2)
=age(1)+2
age(1)
=10
age(2)
=12
age(3)
=14
age(4)
=16
age(5)
=18
回溯阶段
递推阶段


age(5)
=age(4)+2
age(4)
=age(3)+2
age(3)
=age(2)+2
age(2)
=age(1)+2
age(1)
=10
age(2)
=12
age(3)
=14
age(4)
=16
age(5)
=18
结束递归的条件
回溯阶段
递推阶段

```
#include <stdio.h>
int main()
{ int age(int n);
   printf("NO.5,age:%d\n",age(5));
   return 0;
}
int age(int n)
{ int c;
   if(n==1)   c=10;
   else    c=age(n-1)+2;
   return(c);
}
```

```
NO.5,age:18
```


main
age(5)
输出age(5)
18
age函数
n=5
c=age(4)+2
age(5)=18
age函数
n=4
c=age(3)+2
age(4)=16
age函数
n=1
c=10
age(1)=10
age函数
n=2
c=age(1)+2
age(2)=12
age函数
n=3
c=age(2)+2
age(3)=14

例7.7 用递归方法求n！。

- 解题思路：
  - 求n！可以用递推方法：即从1开始，乘2，再乘3……一直乘到n。
  - 递推法的特点是从一个已知的事实(如1!=1)出发，按一定规律推出下一个事实(如2!=1!*2)，再从这个新的已知的事实出发，再向下推出一个新的事实(3!=3*2!)。n!=n*(n-1)!。

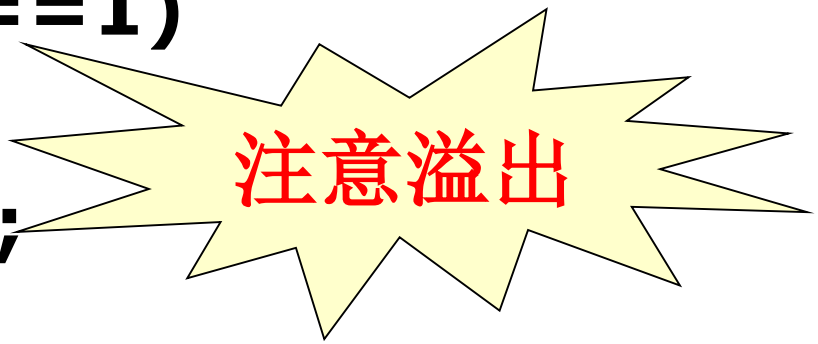
例7.7 用递归方法求n！。

- 解题思路：
  - 求n！也可以用递归方法，即5！等于4！×5，而4！＝3！×4…，1！＝1
  - 可用下面的递归公式表示：

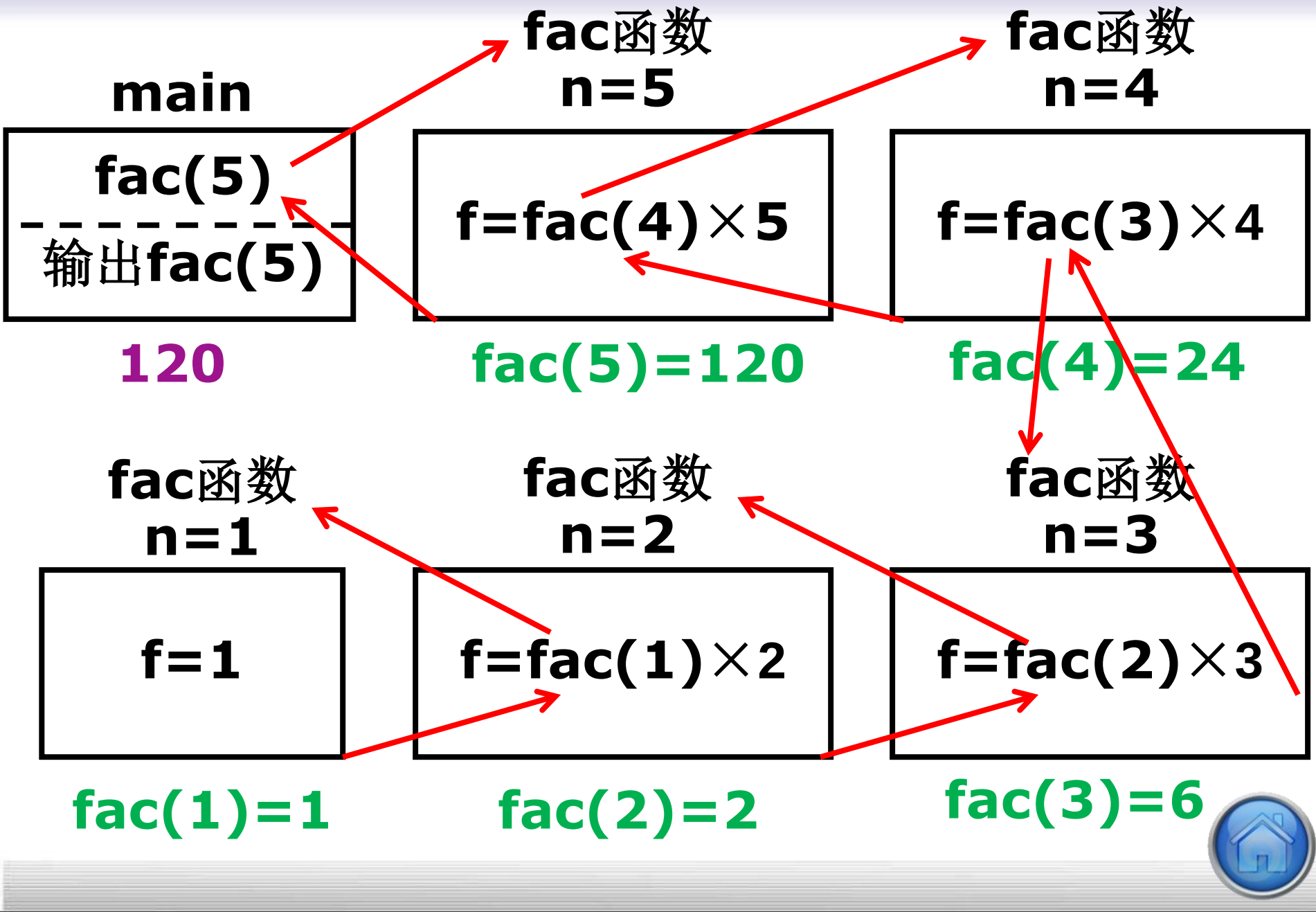
$$n!=\begin{cases} n!=1 & (n=0,1) \\ n\bullet(n-1) & (n>1) \end{cases}$$

```
#include <stdio.h>
int main()
 {int fac(int n);
  int n;  int y;
  printf("input an integer number:");
  scanf("%d",&n);
  y=fac(n);
  printf("%d!=%d\n",n,y);
  return 0;
}
```

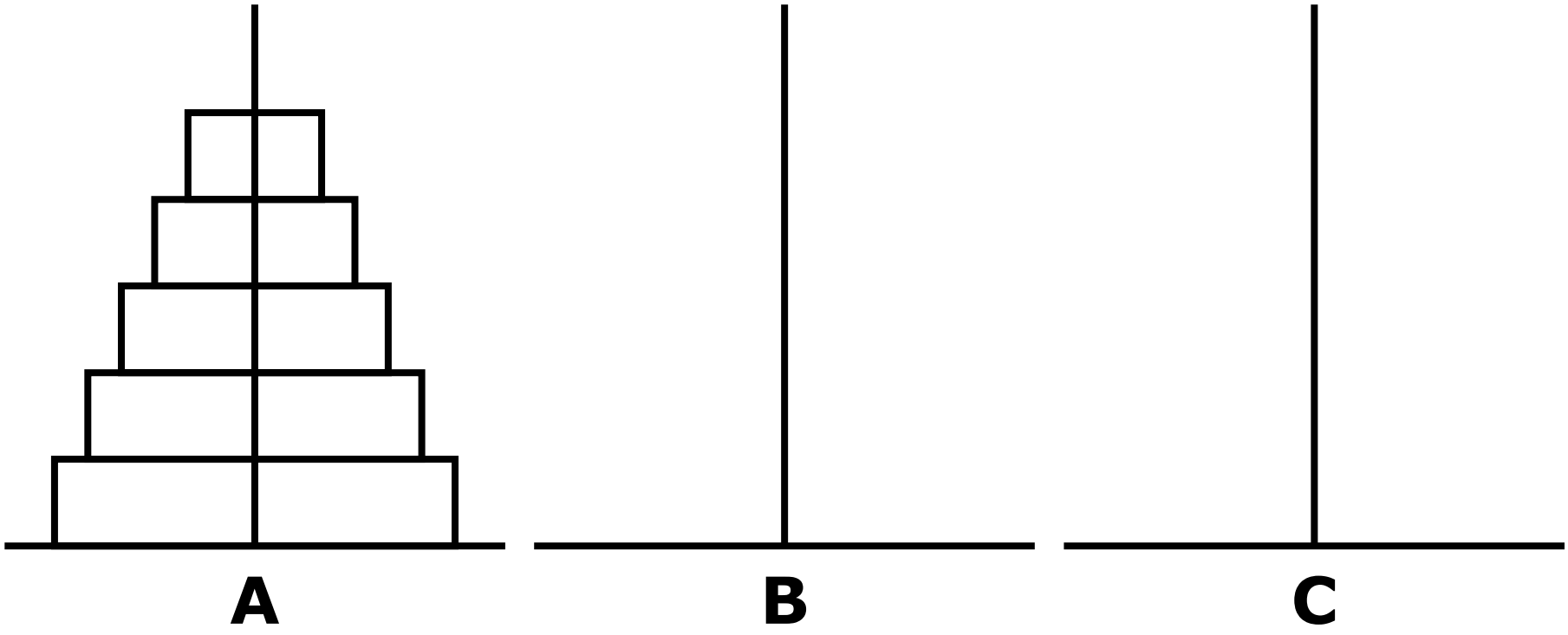
```
int fac(int n)
 {
    int f;
    if(n<0)
       printf("n<0,data error!");
    else if(n==0 | | n==1)
       f=1;
    else  f=fac(n-1)*n;
    return(f);
 }
```




```
input an integer number:31
31!=738197504
```


main
fac(5)
输出fac(5)
120
fac函数
n=5
f=fac(4)×5
fac(5)=120
fac函数
n=4
f=fac(3)×4
fac(4)=24
fac函数
n=3
f=fac(2)×3
fac(3)=6
fac函数
n=2
f=fac(1)×2
fac(2)=2
fac函数
n=1
f=1
fac(1)=1

例7.8 Hanoi（汉诺）塔问题。古代有一个梵塔，塔内有3个座A、B、C，开始时A座上有64个盘子，盘子大小不等，大的在下，小的在上。有一个老和尚想把这64个盘子从A座移到C座，但规定每次只允许移动一个盘，且在移动过程中在3个座上都始终保持大盘在下，小盘在上。在移动过程中可以利用B座。要求编程序输出移动一盘子的步骤。

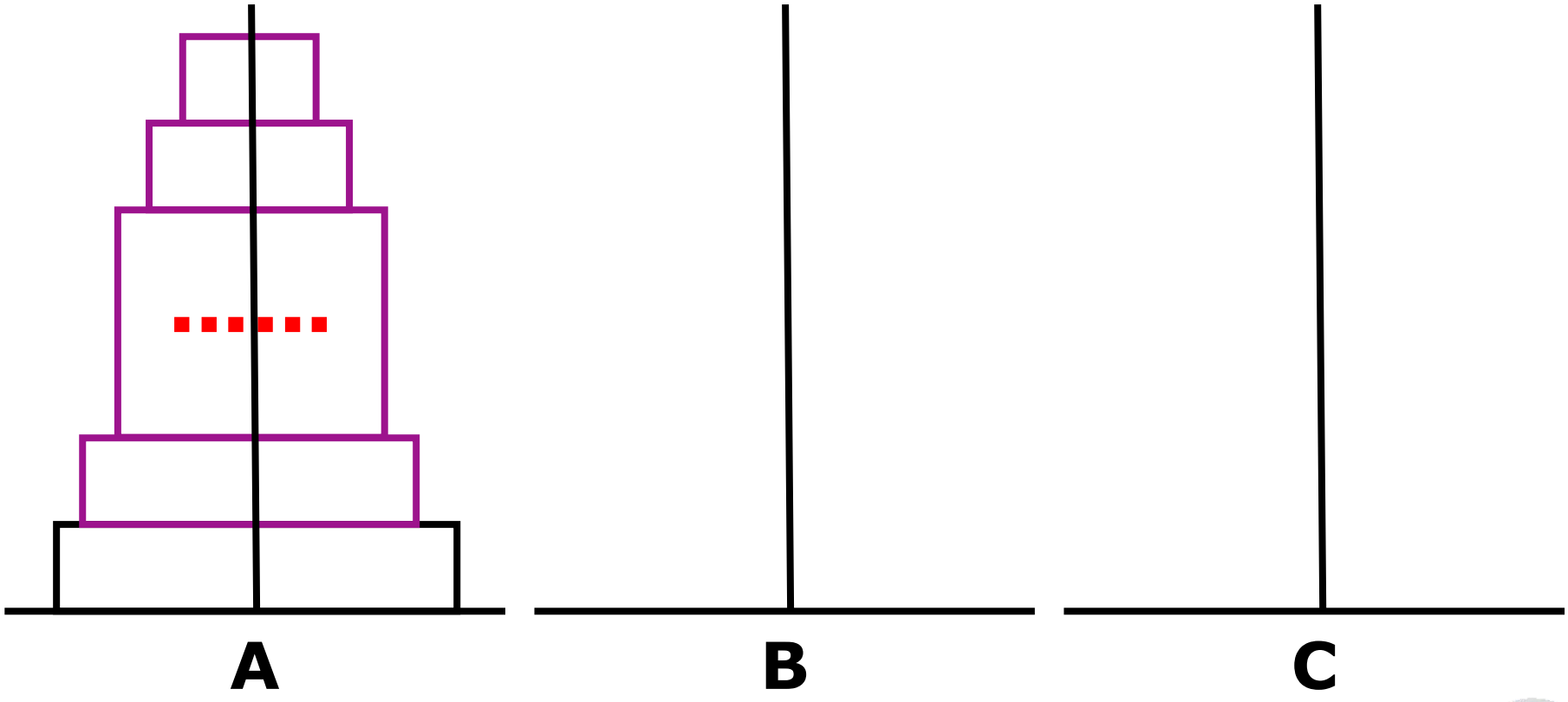

A
B
C

- 解题思路：
  - 要把64个盘子从A座移动到C座，需要移动大约2^{64} 次盘子。一般人是不可能直接确定移动盘子的每一个具体步骤的
  - 老和尚会这样想：假如有另外一个和尚能有办法将上面63个盘子从一个座移到另一座。那么，问题就解决了。此时老和尚只需这样做：

- 解题思路：
  - (1) 命令第2个和尚将63个盘子从A座移到B座
  - (2) 自己将1个盘子（最底下的、最大的盘子）从A座移到C座
  - (3) 再命令第2个和尚将63个盘子从B座移到C座

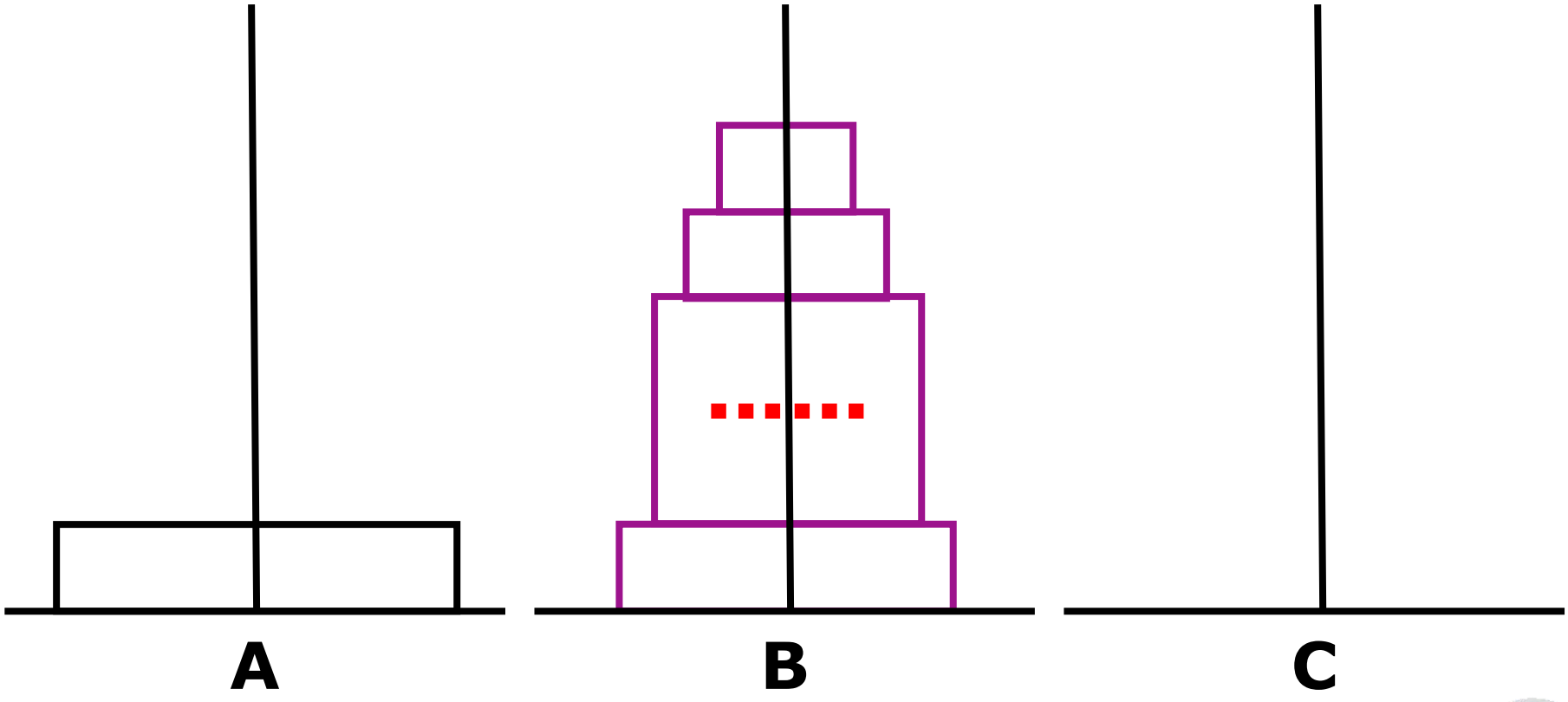
第1个和尚的做法

将63个从A到B



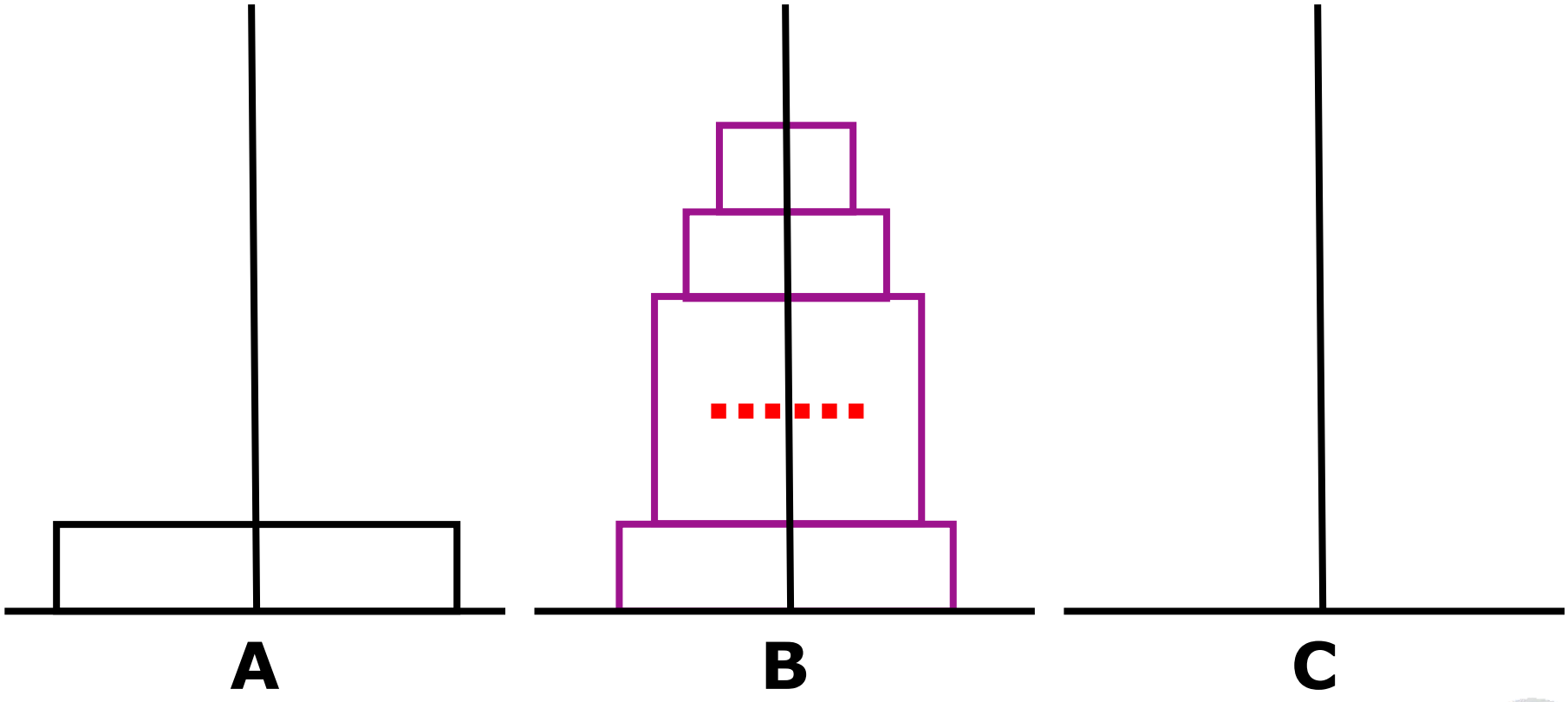
第1个和尚的做法

将63个从A到B



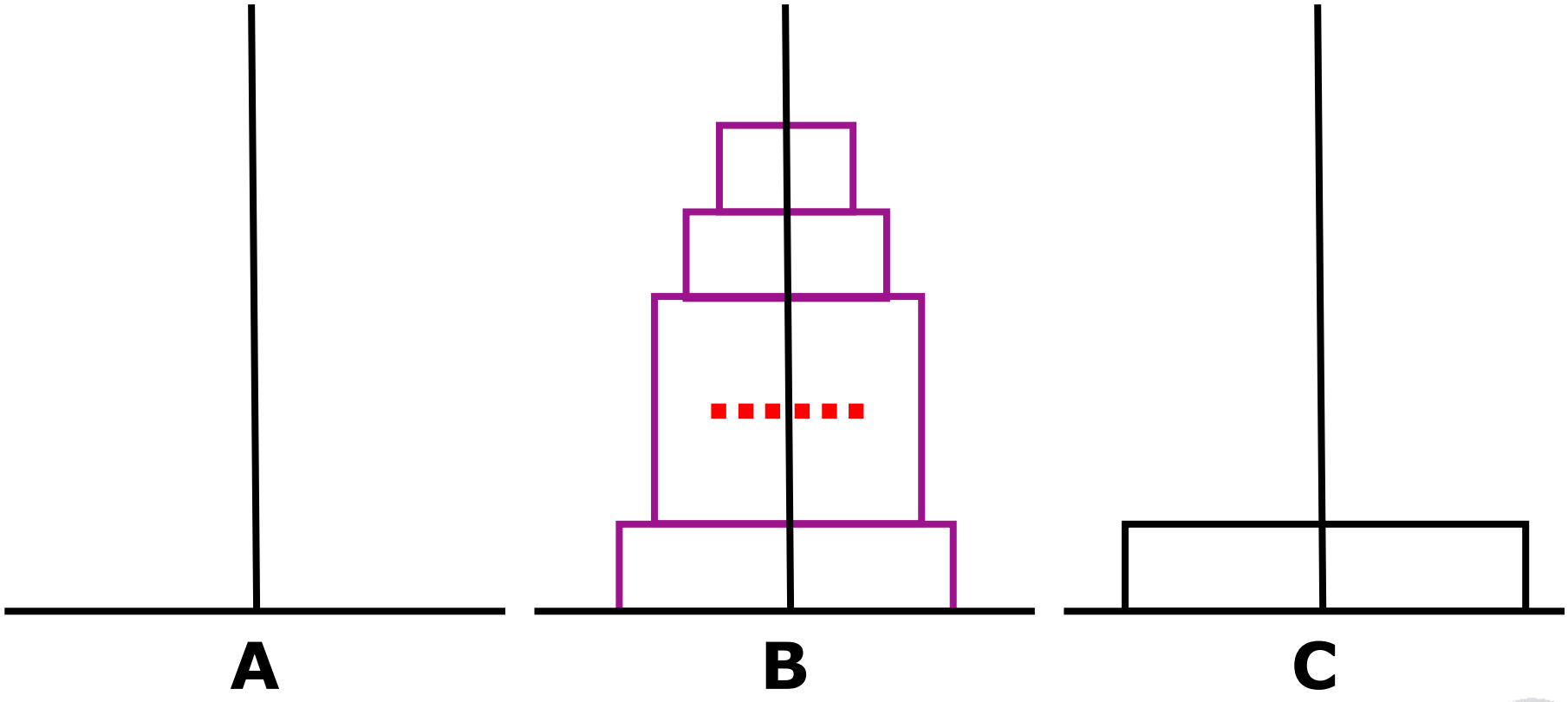
第1个和尚的做法

将1个从A到C



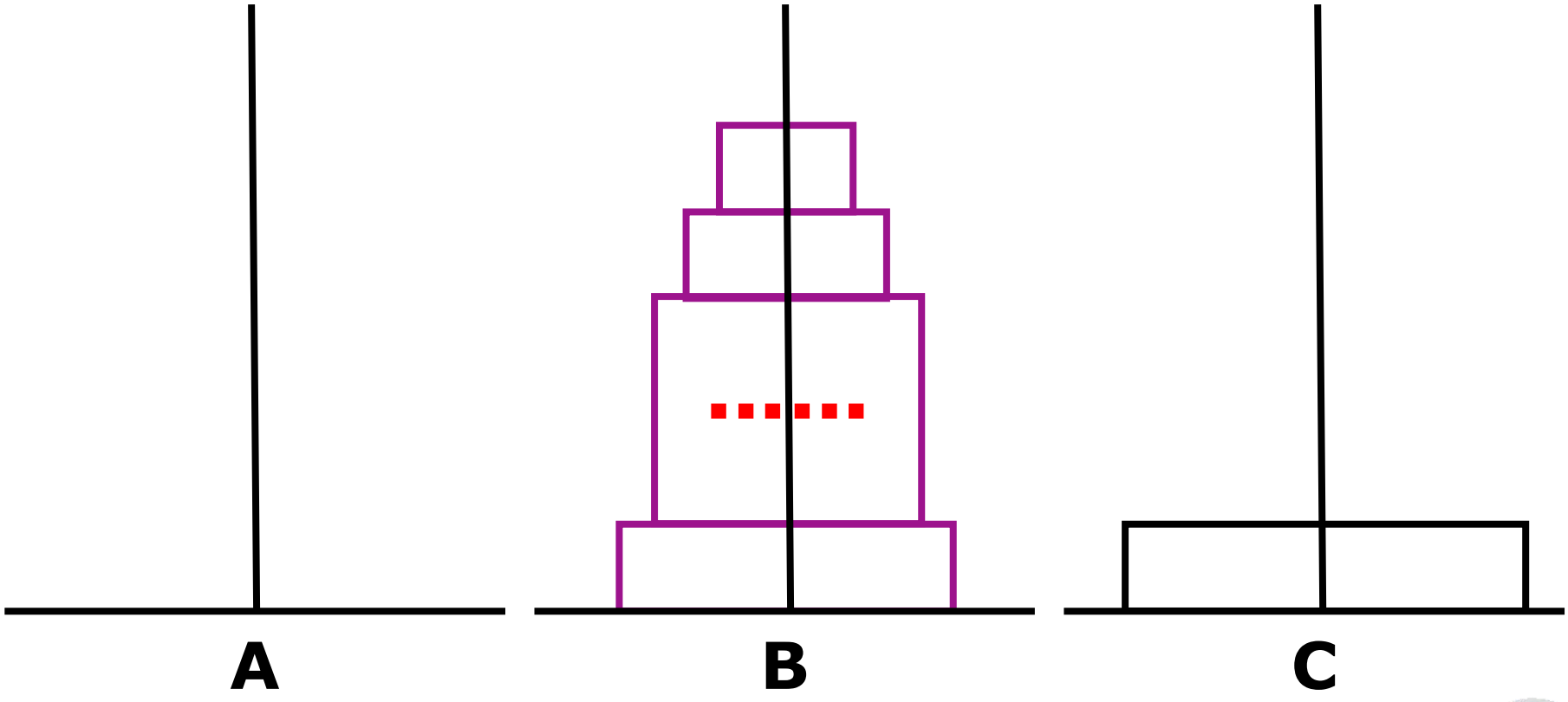
第1个和尚的做法

将1个从A到C



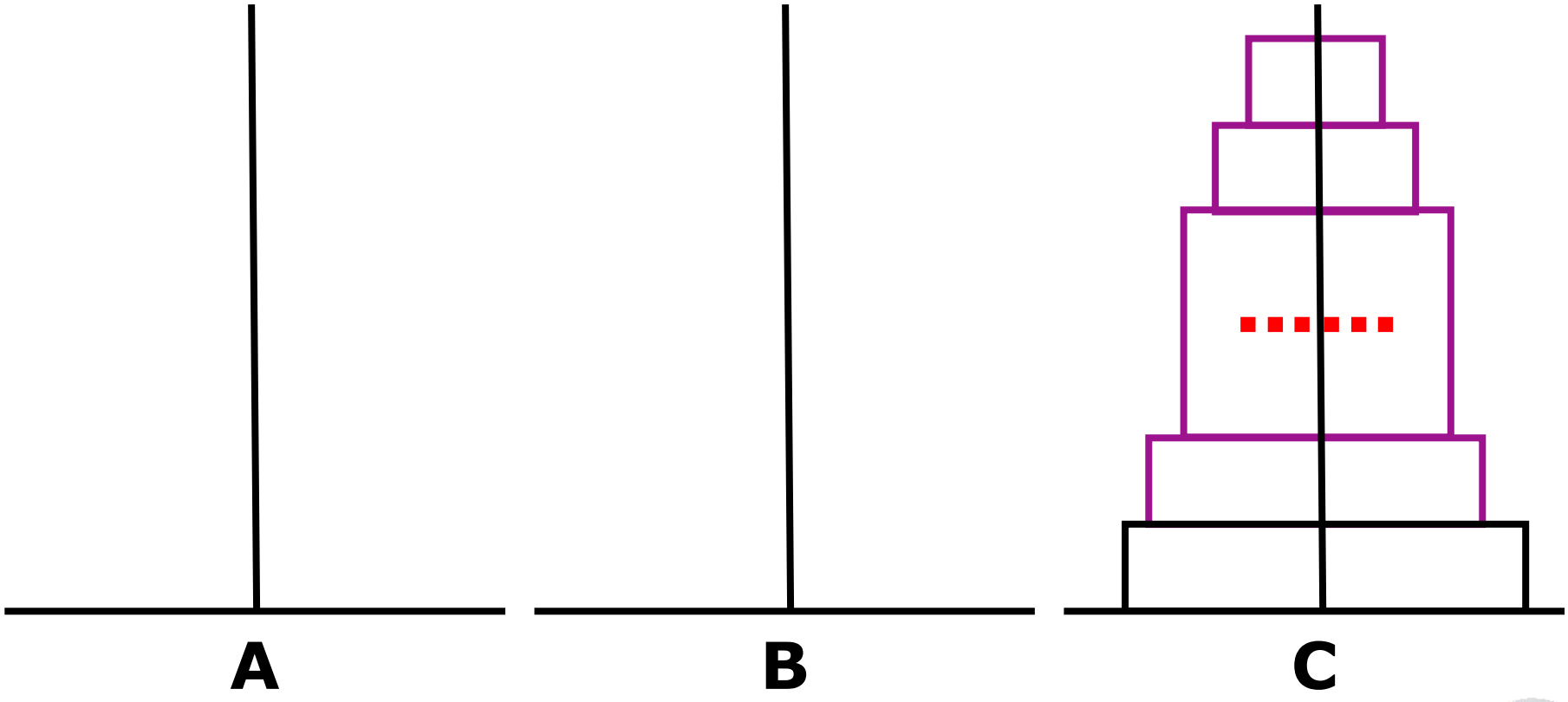
第1个和尚的做法

将63个从B到C



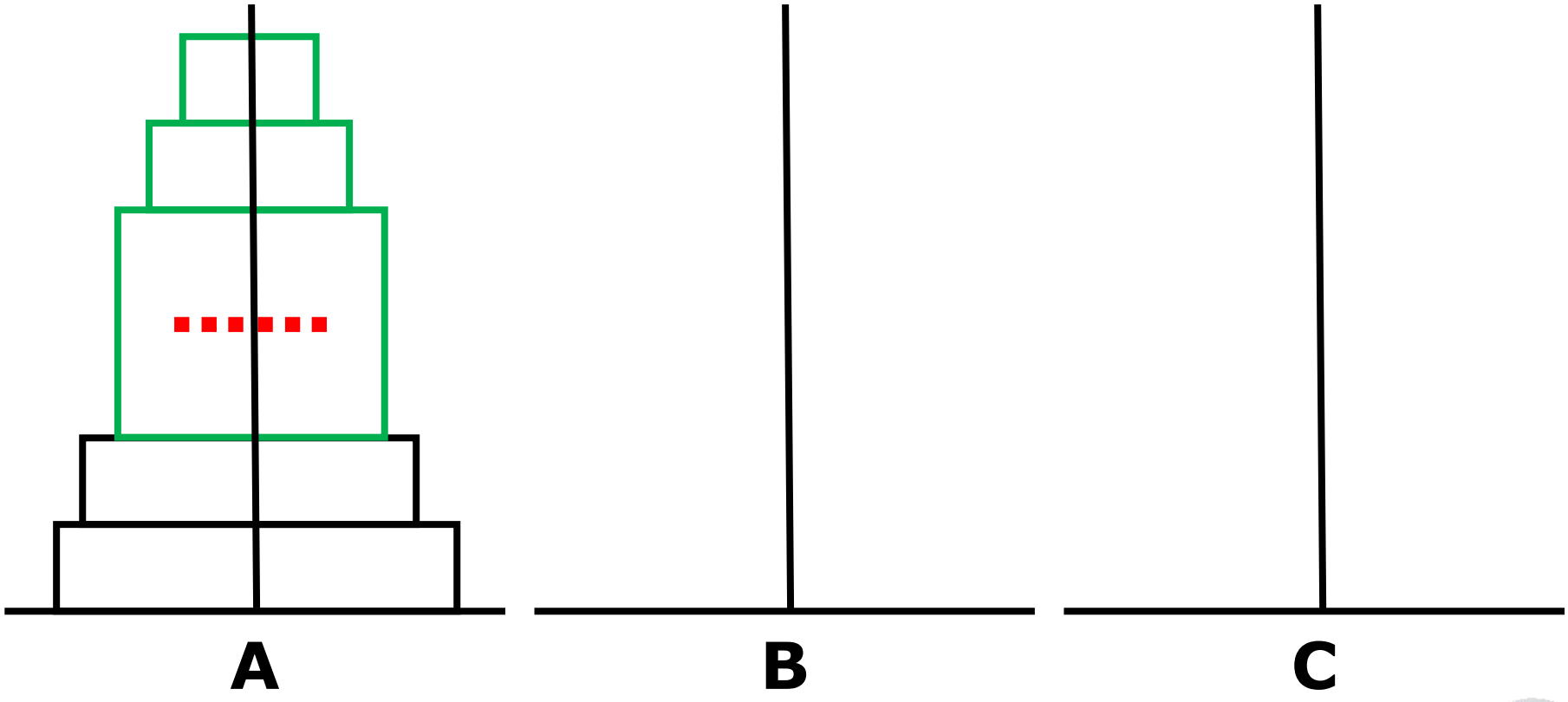
第1个和尚的做法

将63个从B到C



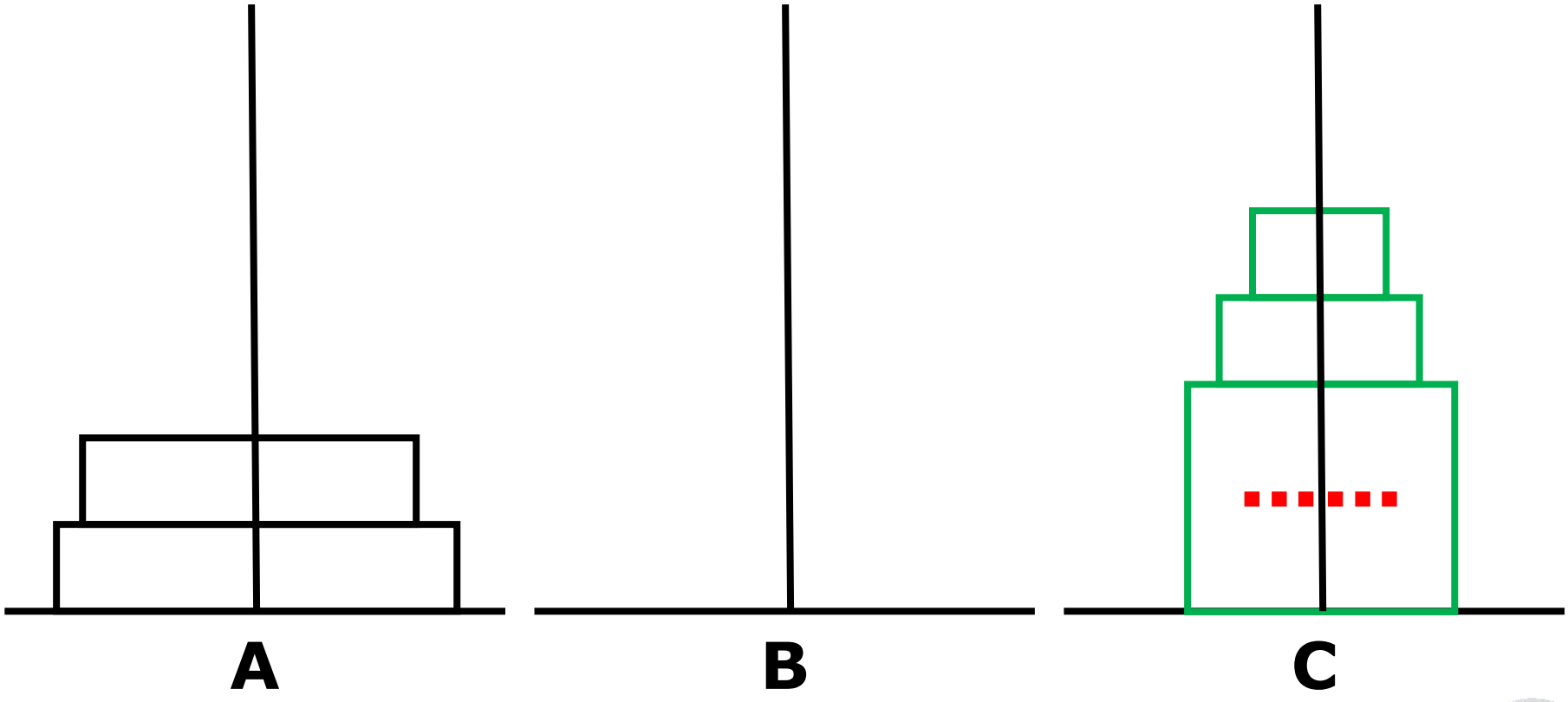
第2个和尚的做法

将62个从A到C



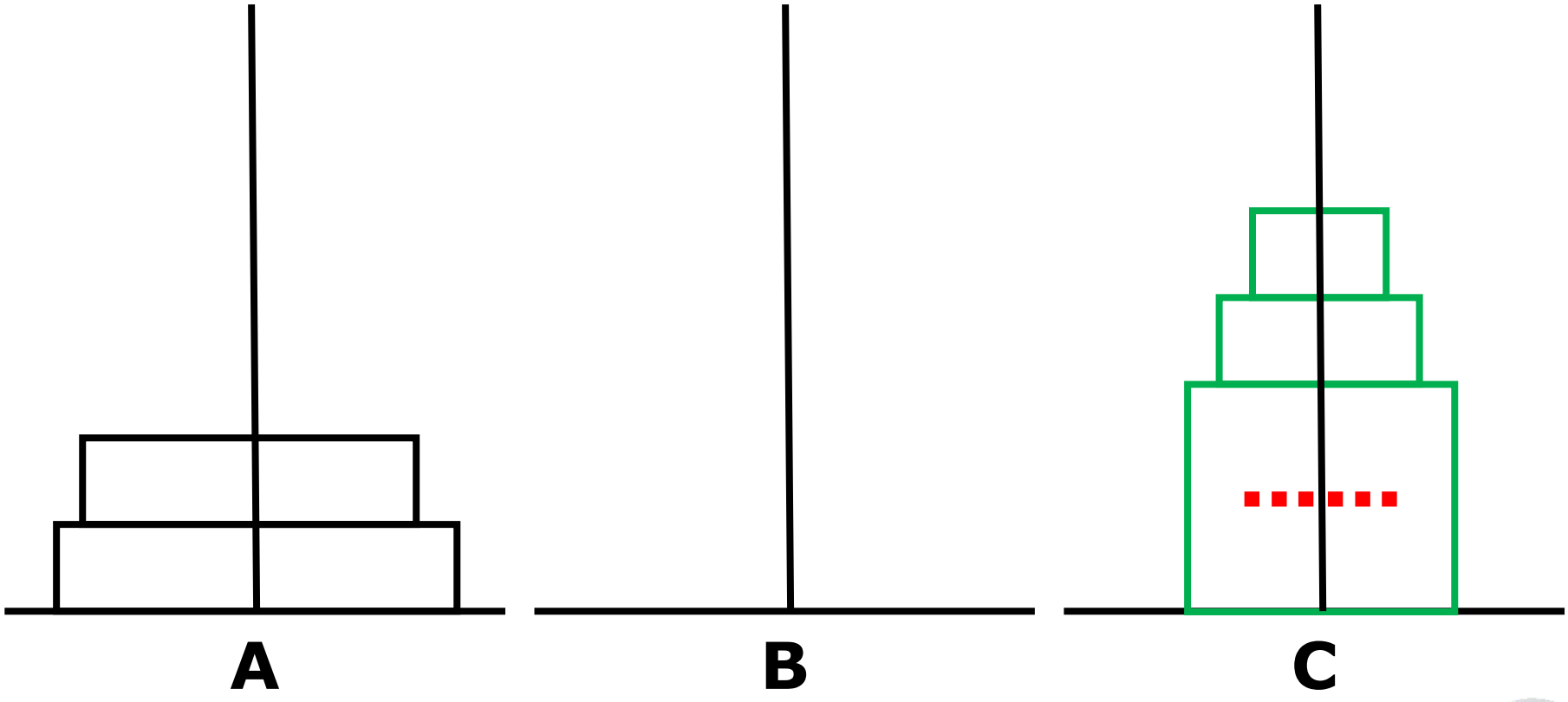
第2个和尚的做法

将62个从A到C



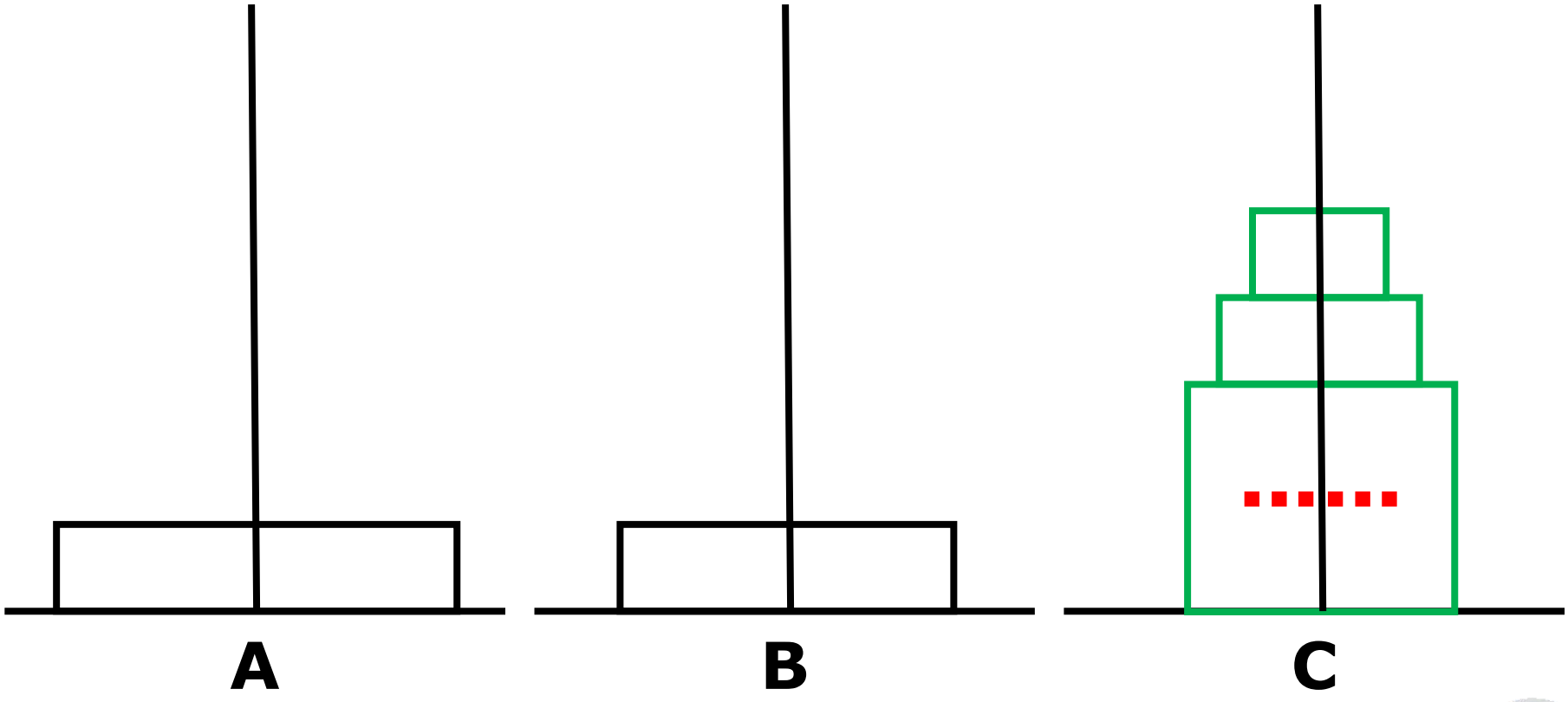
第2个和尚的做法

将1个从A到B



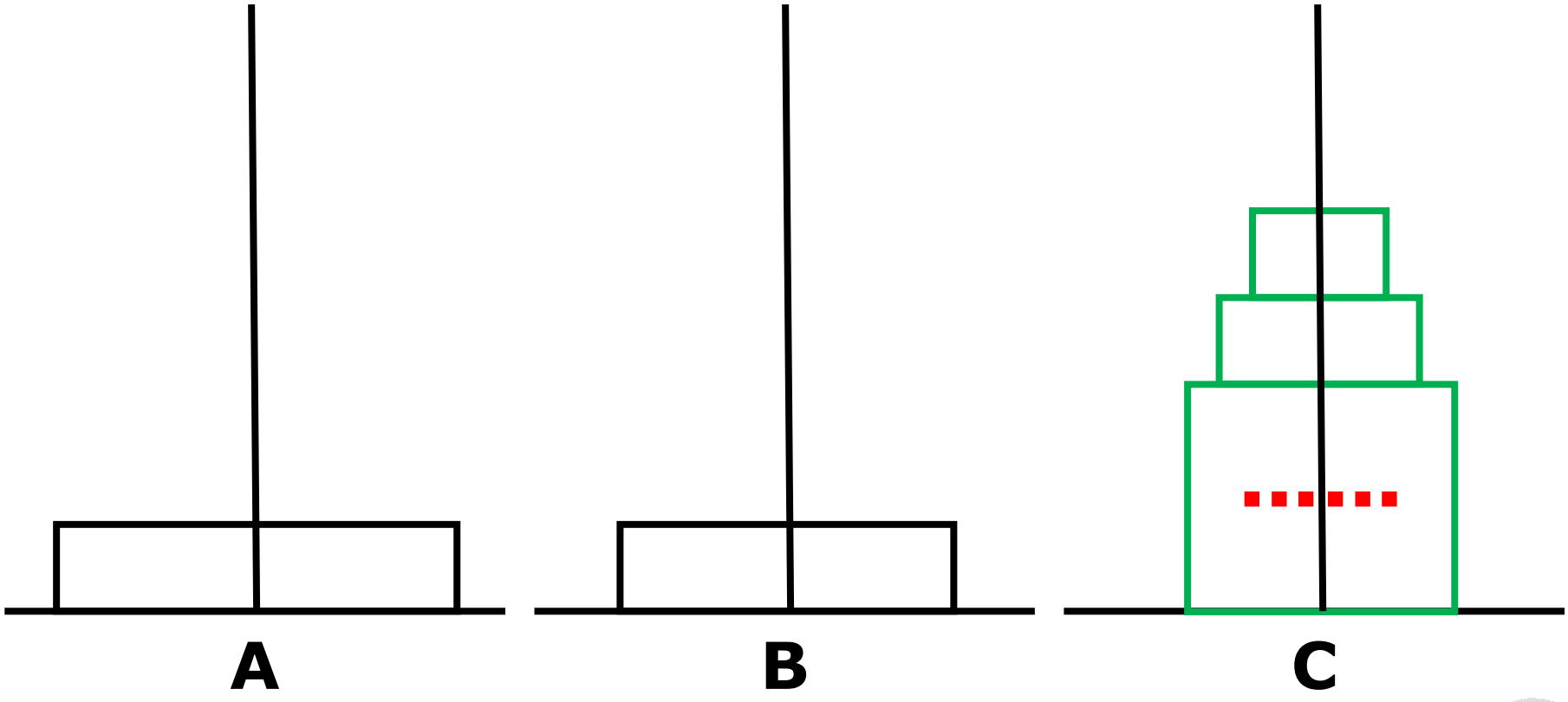
第2个和尚的做法

将1个从A到B



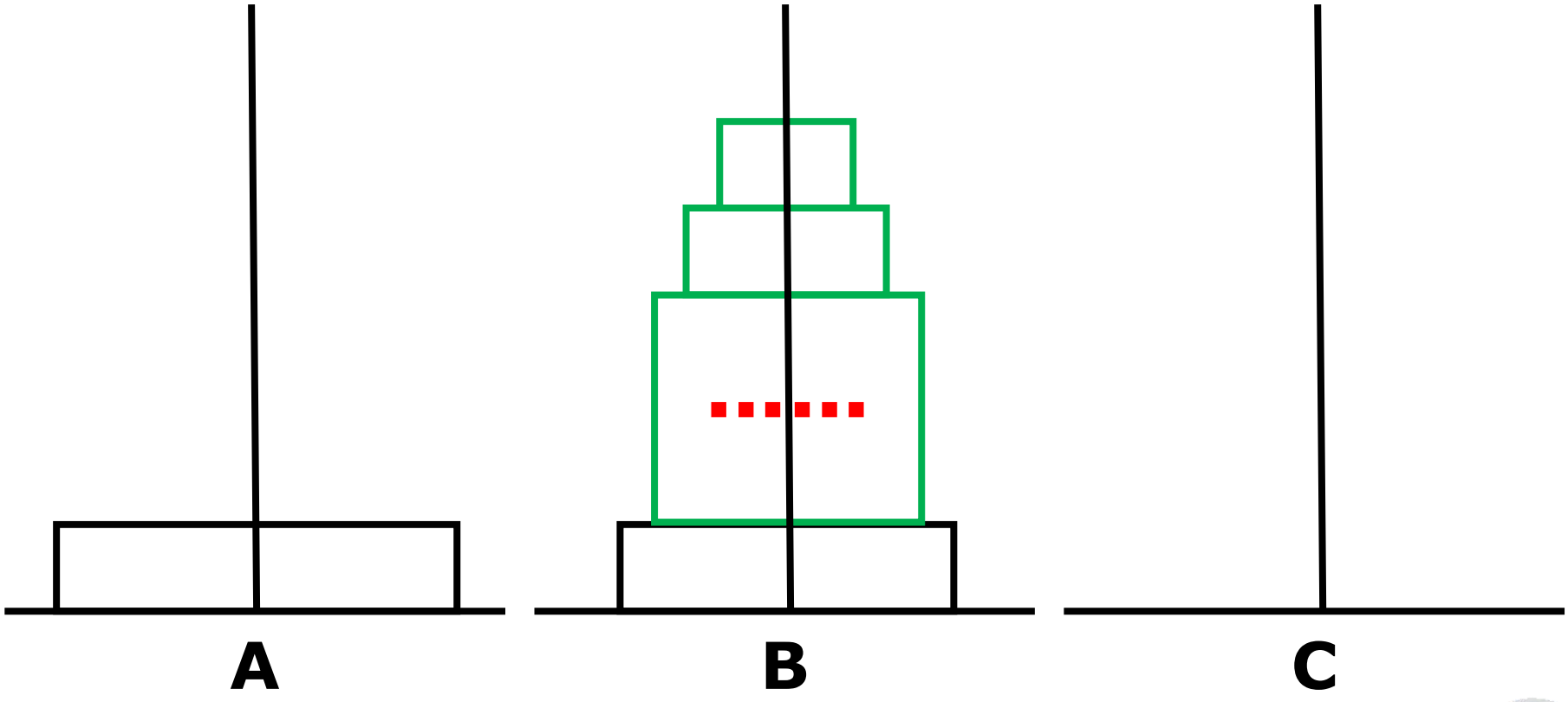
第2个和尚的做法

将62个从C到B



第2个和尚的做法

将62个从C到B



第3个和尚的做法

第4个和尚的做法

第5个和尚的做法

第6个和尚的做法

第7个和尚的做法

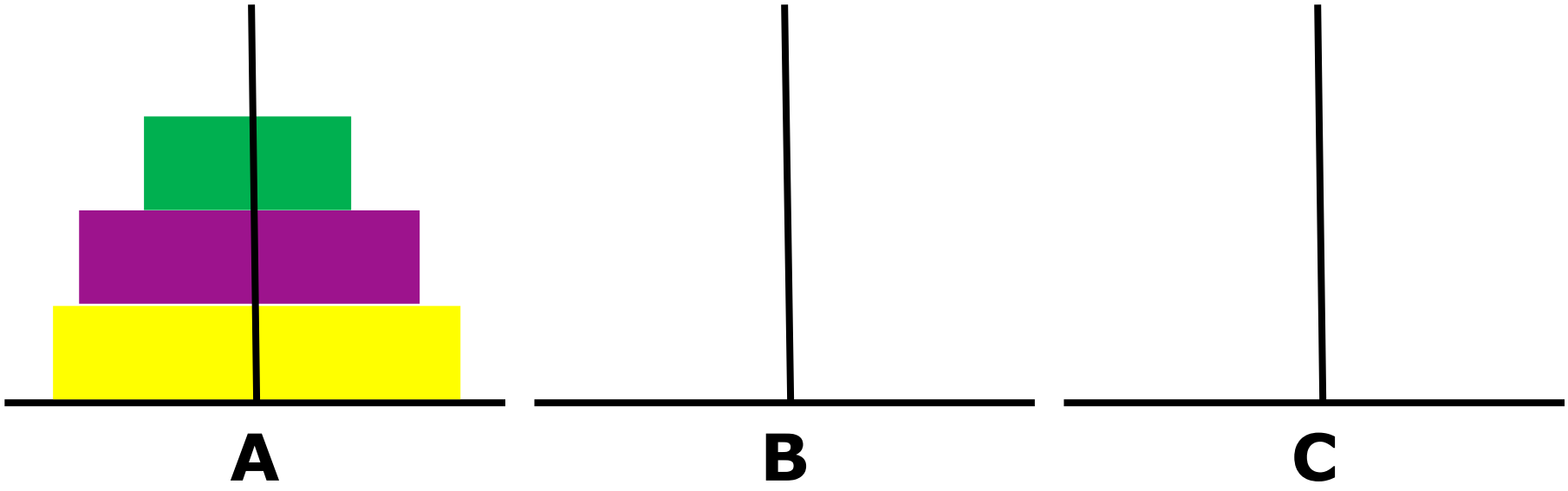
……

第63个和尚的做法

第64个和尚仅做：将1个从A移到C

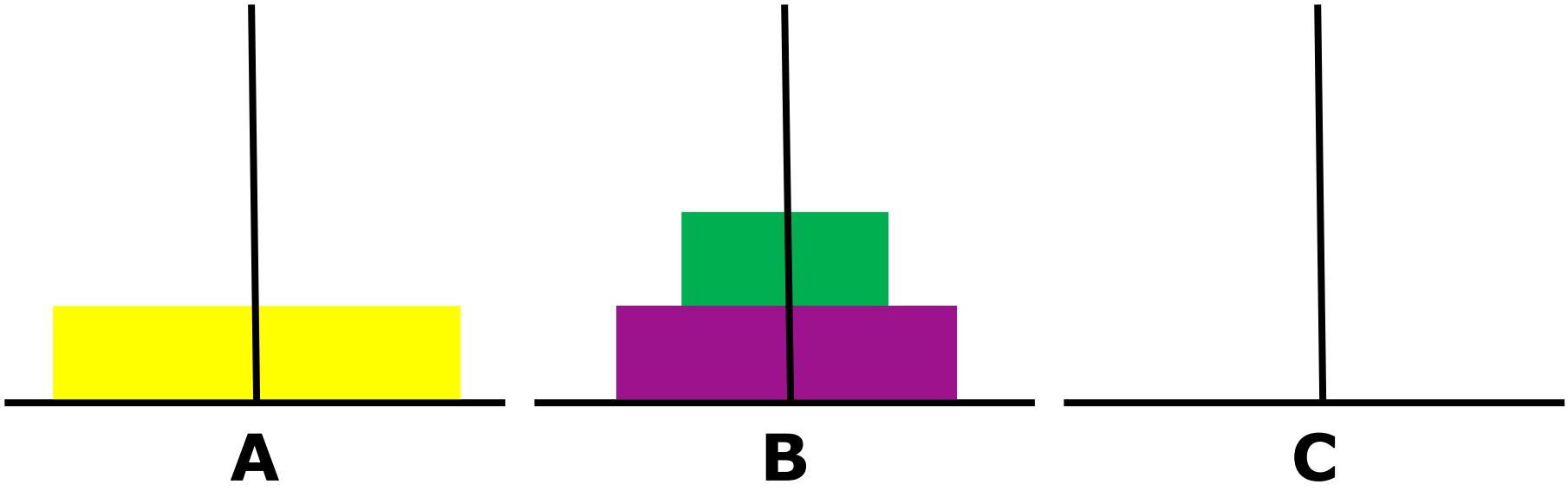
将3个盘子从A移到C的全过程

将2个盘子从A移到B



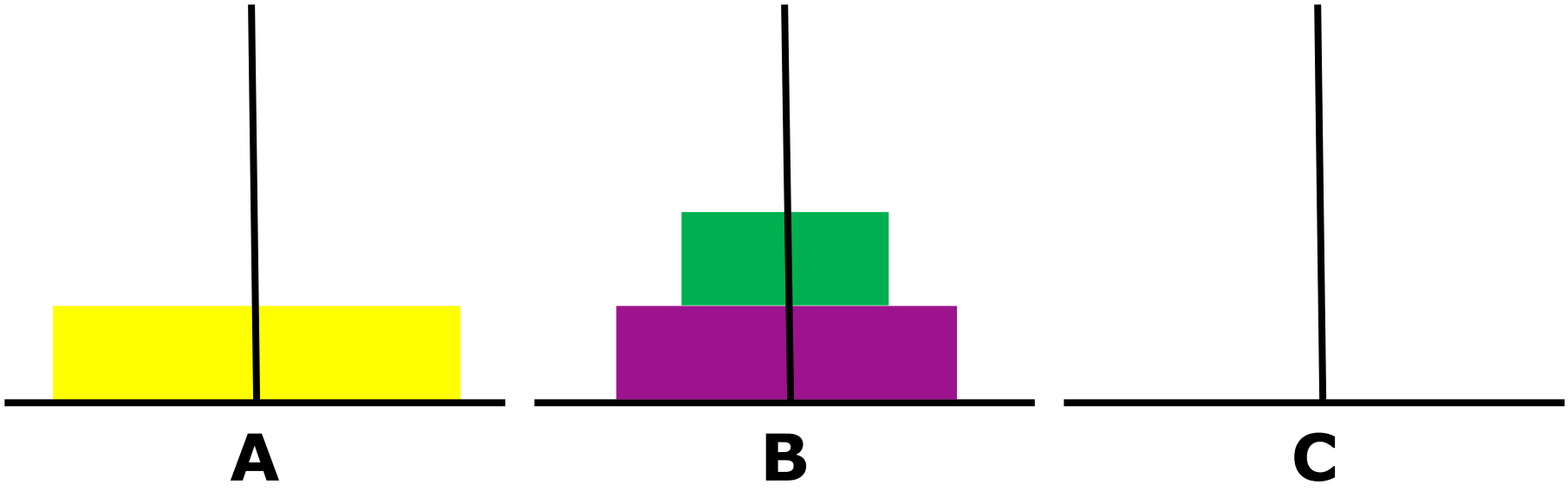
将3个盘子从A移到C的全过程

将2个盘子从A移到B



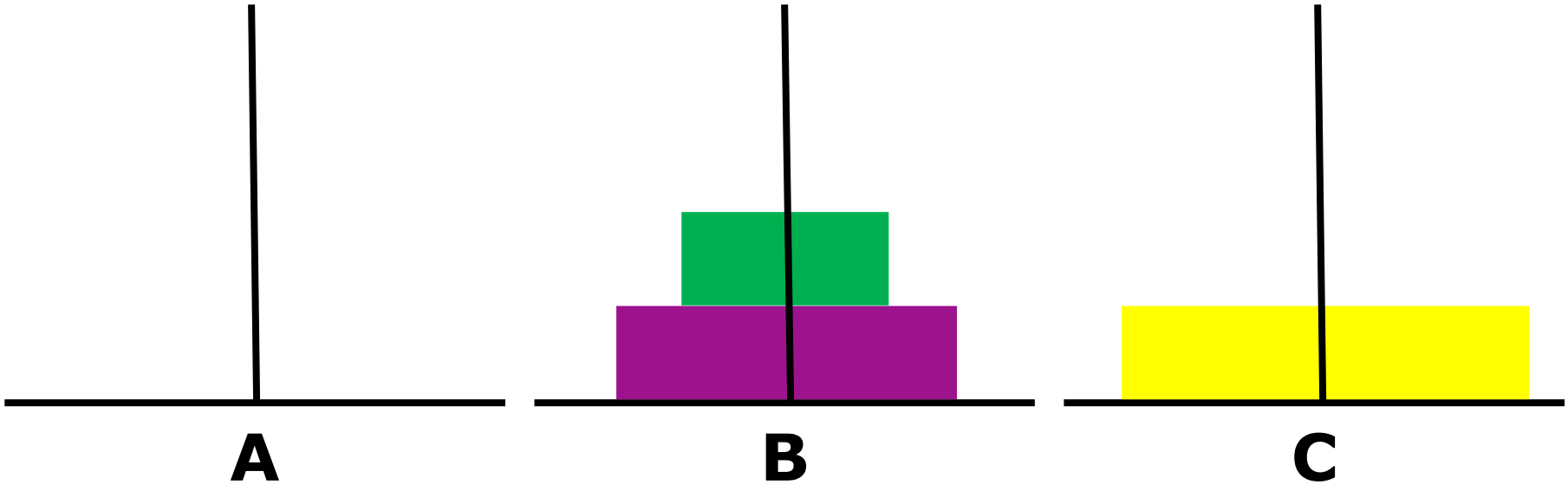
将3个盘子从A移到C的全过程

将1个盘子从A移到C



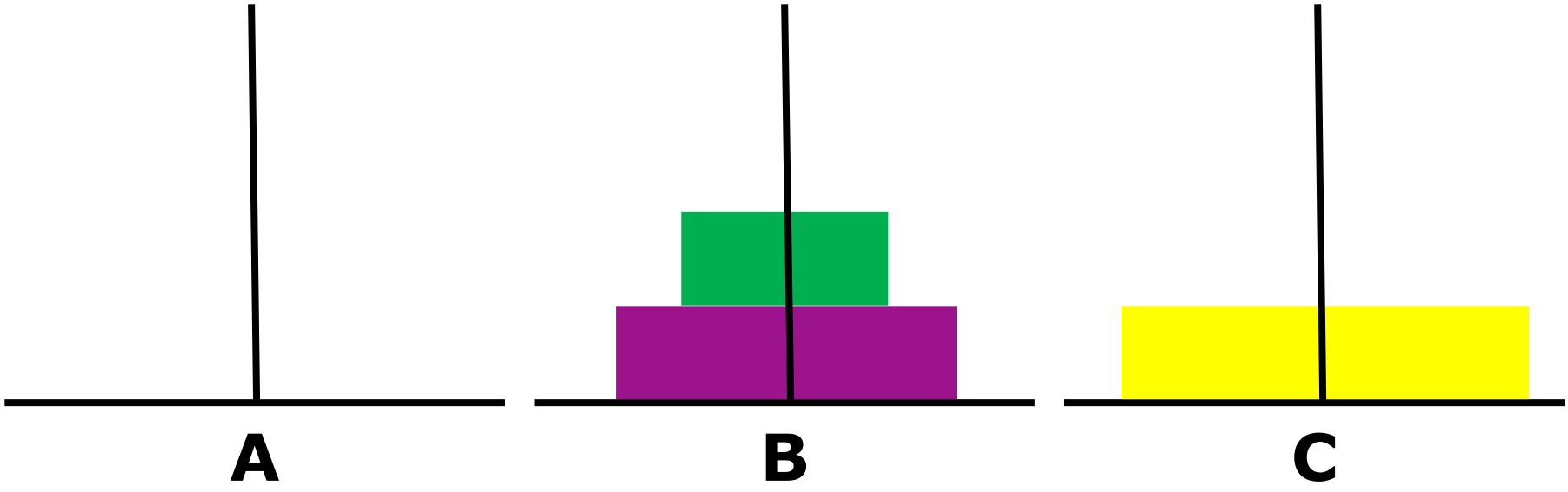
将3个盘子从A移到C的全过程

将1个盘子从A移到C



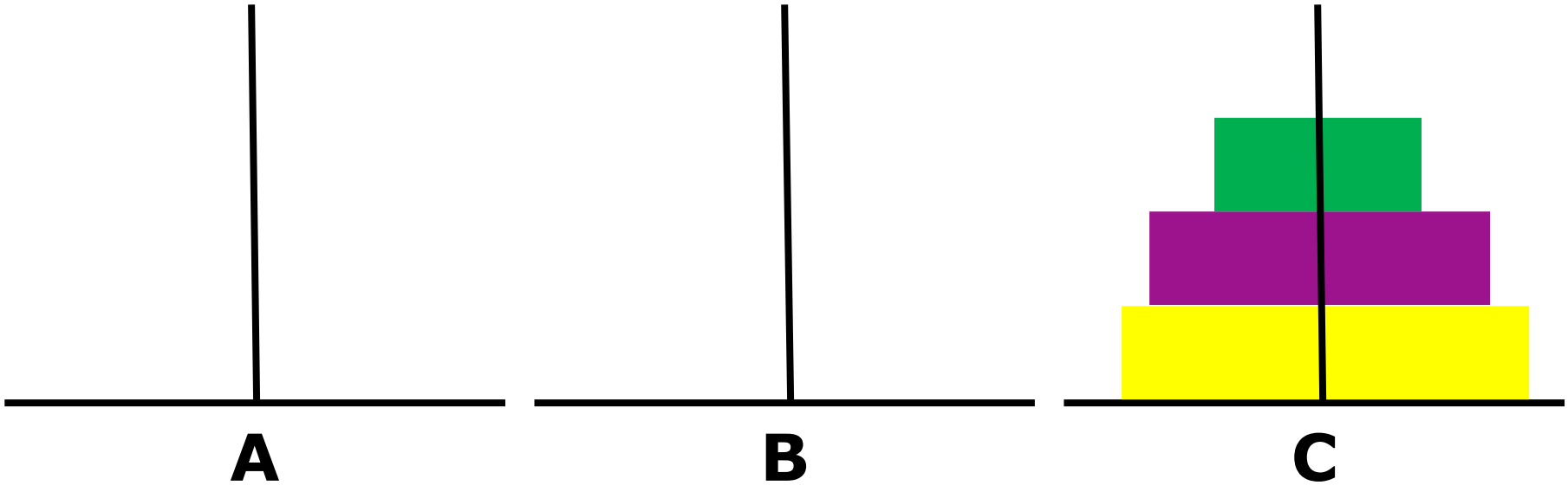
将3个盘子从A移到C的全过程

将2个盘子从B移到C



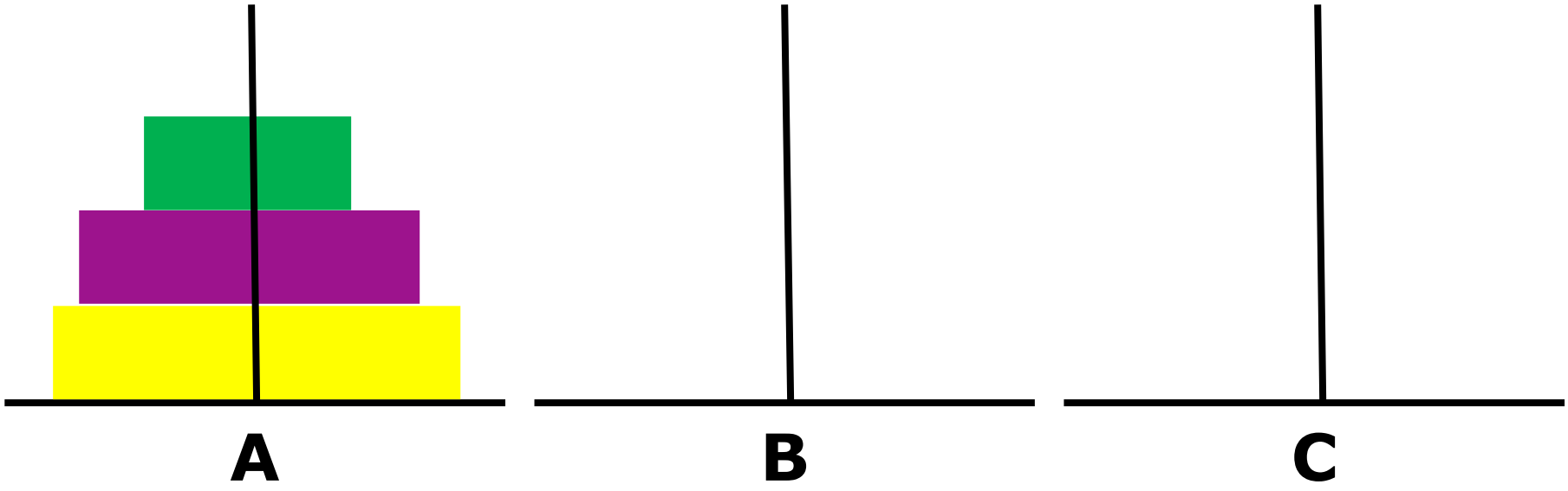
将3个盘子从A移到C的全过程

将2个盘子从B移到C



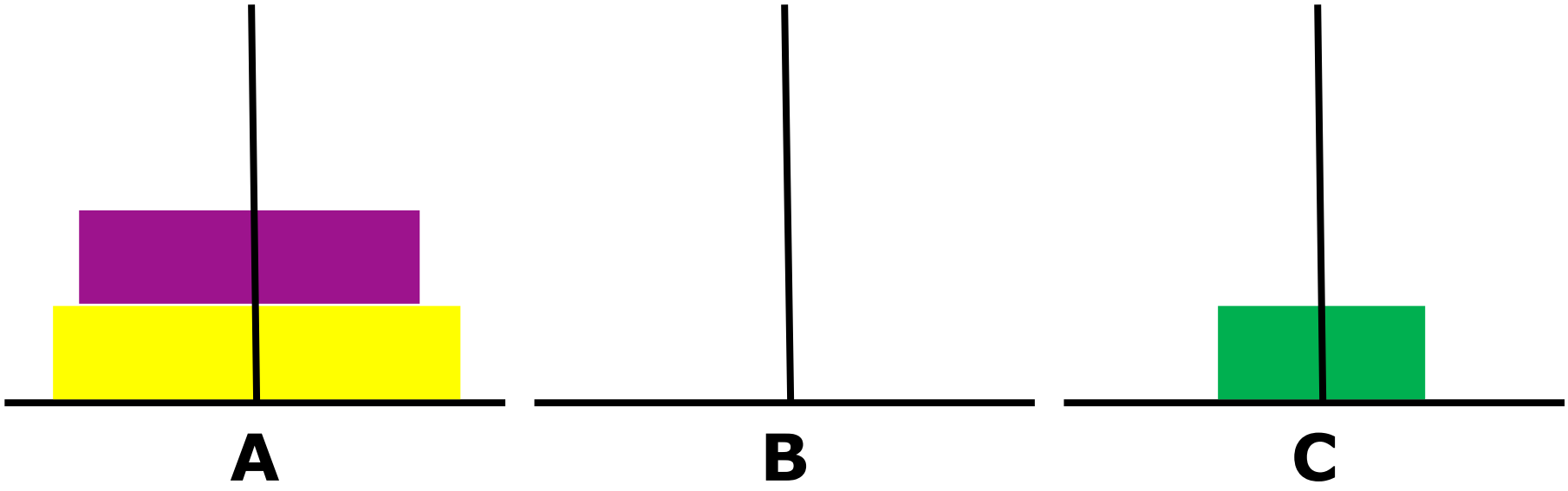
将2个盘子从A移到B的过程

将1个盘子从A移到C



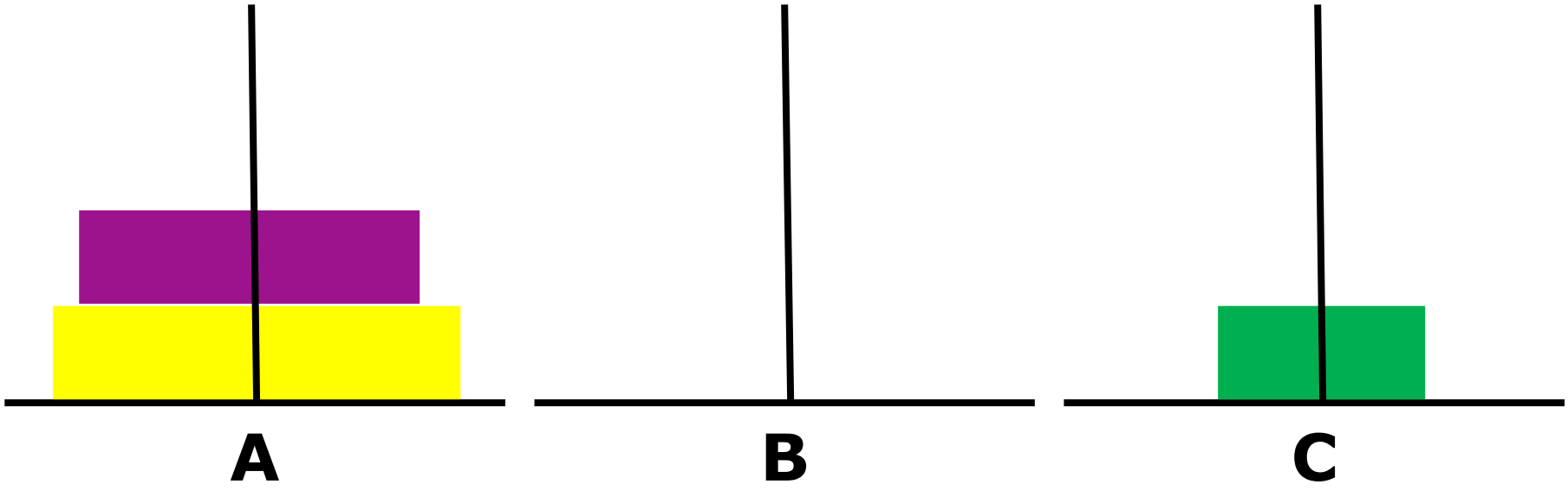
将2个盘子从A移到B的过程

将1个盘子从A移到C



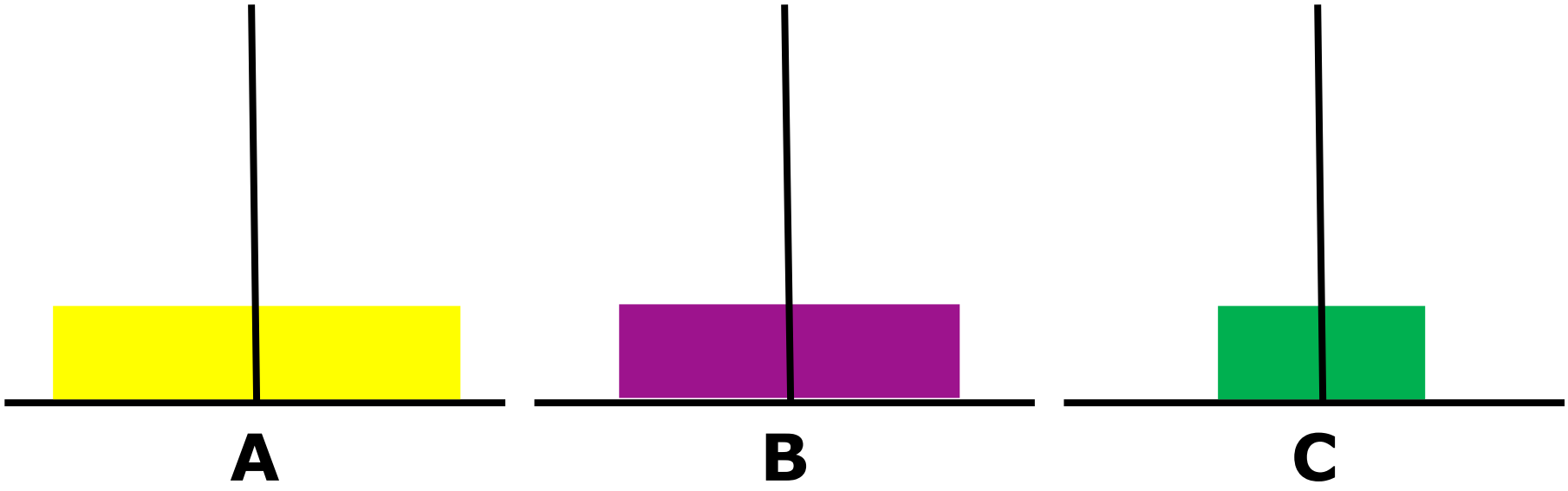
将2个盘子从A移到B的过程

将1个盘子从A移到B



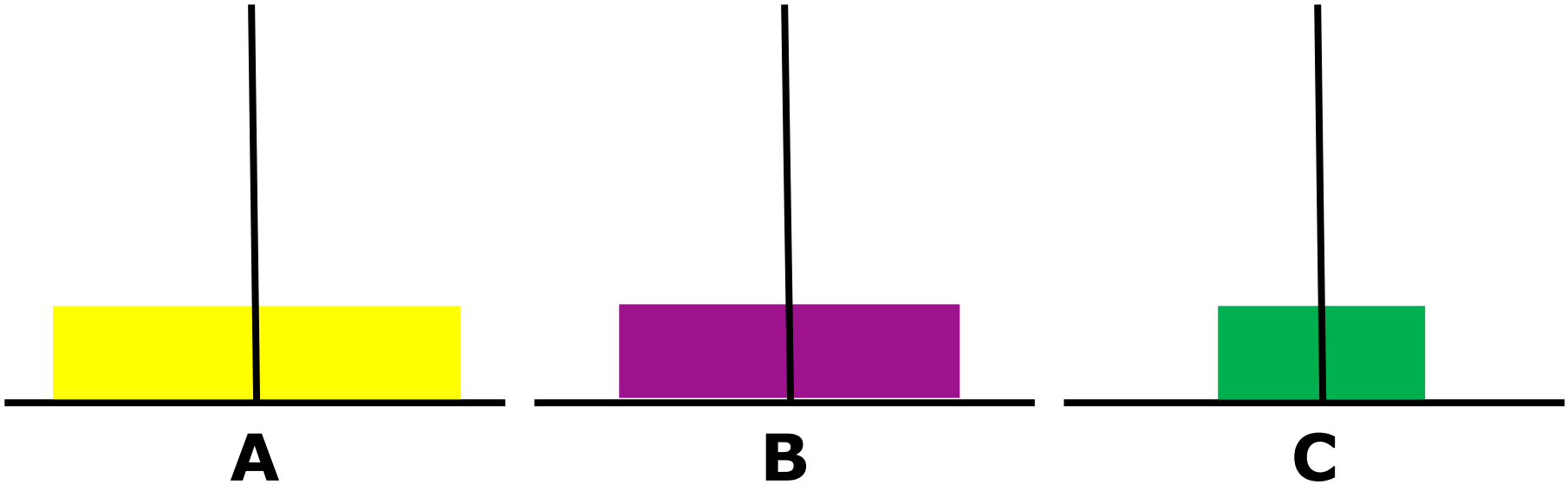
将2个盘子从A移到B的过程

将1个盘子从A移到B



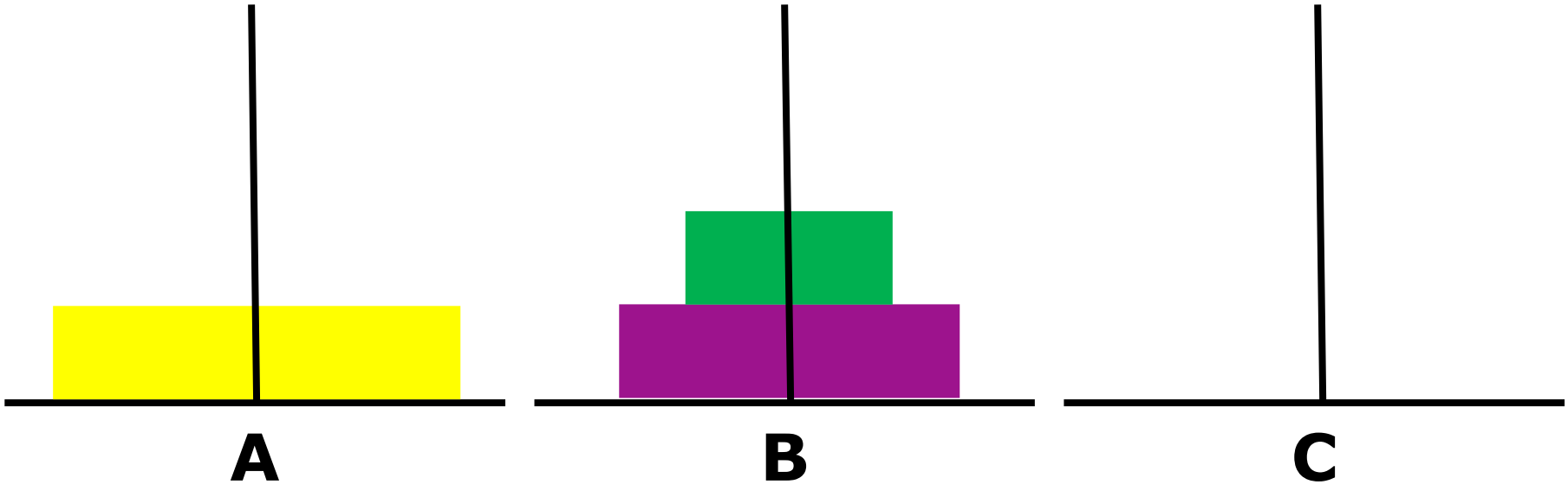
将2个盘子从A移到B的过程

将1个盘子从C移到B

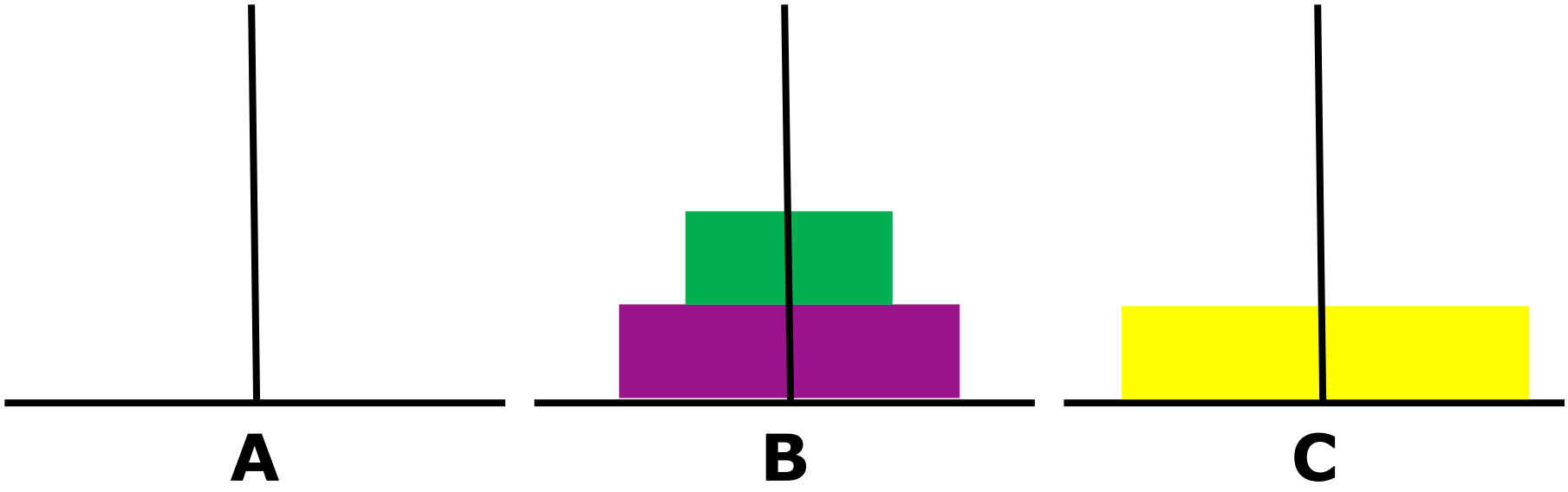


将2个盘子从A移到B的过程

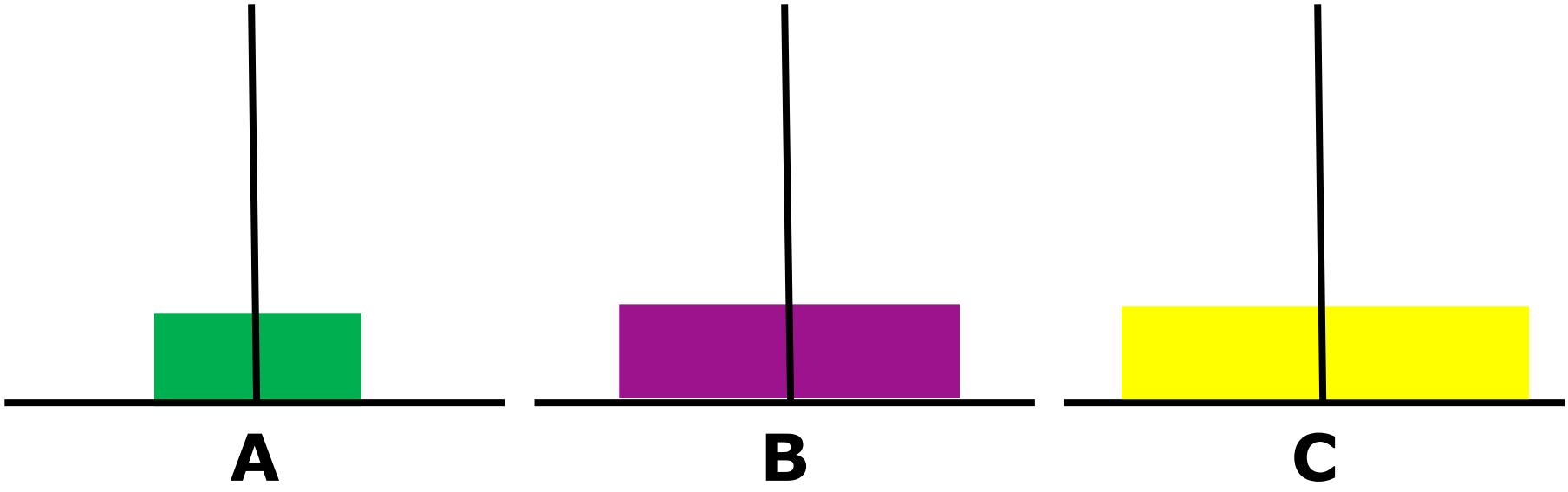
将1个盘子从C移到B



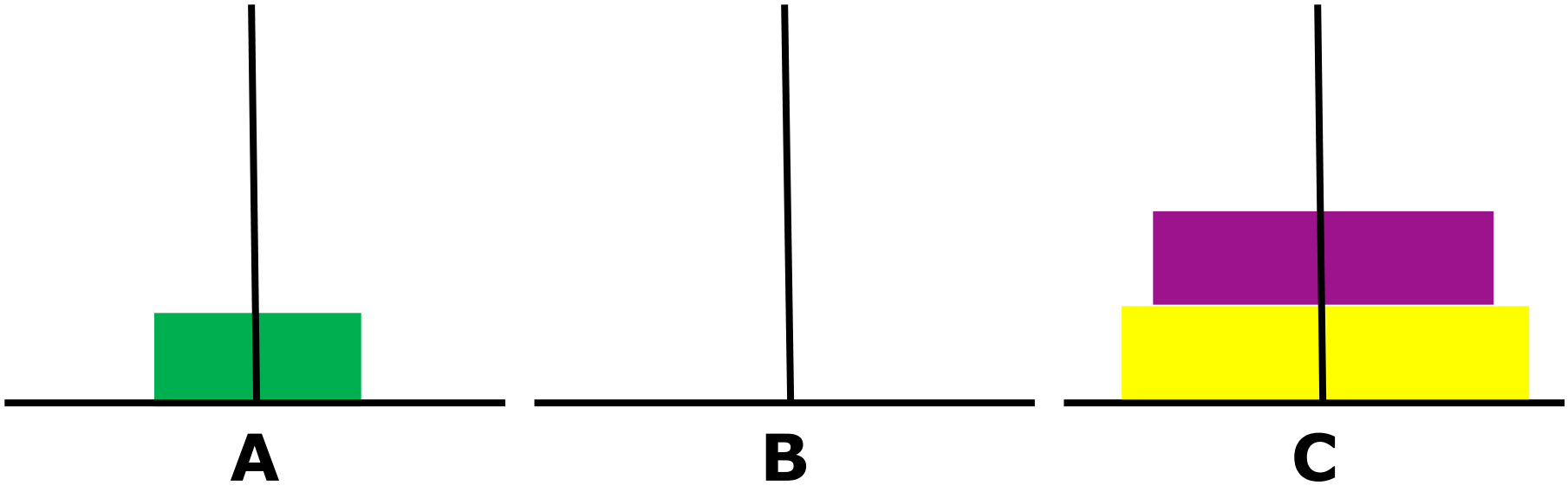
将2个盘子从B移到C的过程



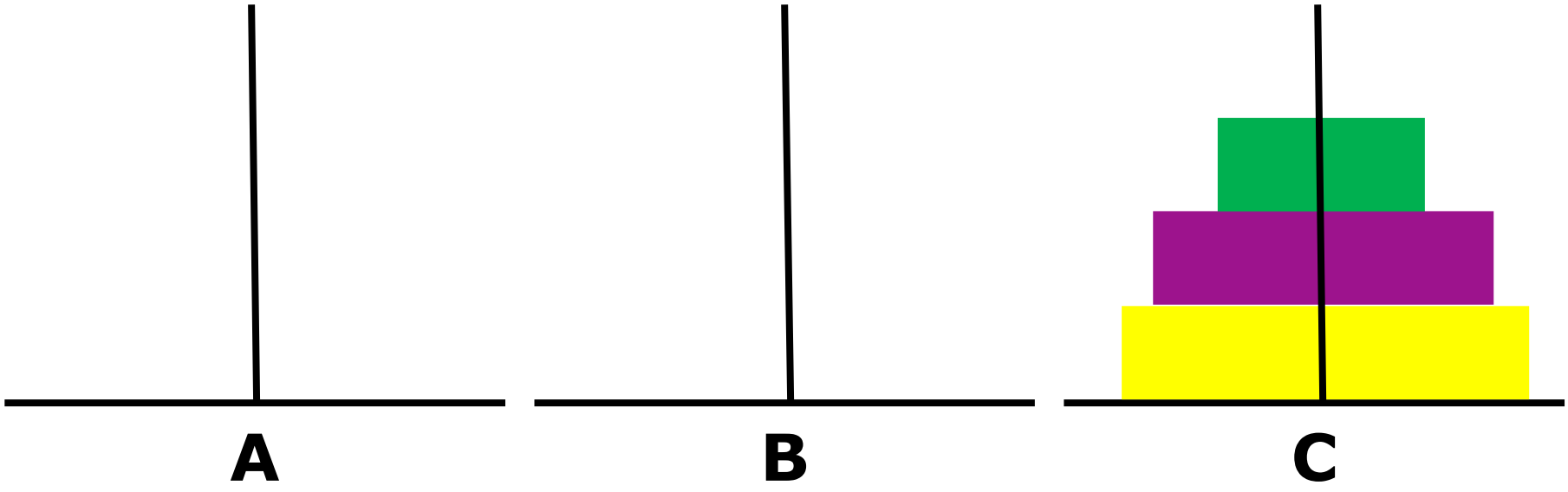
将2个盘子从B移到C的过程



将2个盘子从B移到C的过程



将2个盘子从B移到C的过程



- 由上面的分析可知：将n个盘子从A座移到C座可以分解为以下3个步骤：
  - (1) 将A上n-1个盘借助C座先移到B座上
  - (2) 把A座上剩下的一个盘移到C座上
  - (3) 将n-1个盘从B座借助于A座移到C座上

- 可以将第(1)步和第(3)步表示为：
  - 将“one”座上n-1个盘移到“two”座(借助“three”座)。
  - 在第(1)步和第(3)步中，one 、two、three和A、B、C的对应关系不同。
  - 对第(1)步，对应关系是one对应A，two对应B，three对应C。
  - 对第(3)步，对应关系是one对应B，two对应C，three对应A。

- 把上面3个步骤分成两类操作：

  (1) 将n-1个盘从一个座移到另一个座上（n＞1）。这就是大和尚让小和尚做的工作，它是一个递归的过程，即和尚将任务层层下放，直到第64个和尚为止。

  (2) 将1个盘子从一个座上移到另一座上。这是大和尚自己做的工作。

- 编写程序。
  - 用**hanoi**函数实现第**1**类操作（即模拟小和尚的任务）
  - 用**move**函数实现第**2**类操作（模拟大和尚自己移盘）
  - 函数调用**hanoi(n,one,two.three)**表示将**n**个盘子从“**one**”座移到“**three**”座的过程**(**借助“**two**”座**)**
  - 函数调用**move(x,y)**表示将**1**个盘子从**x** 座移到**y** 座的过程。**x**和**y**是代表**A**、**B**、**C**座之一，根据每次不同情况分别取**A**、**B**、**C**代入

```
#include <stdio.h>
int main()
{  void hanoi(int n,char one,
                    char two,char three);
   int m;
   printf(“the number of diskes:");
   scanf("%d",&m);
   printf("move %d diskes:\n",m);
   hanoi(m,'A','B','C');
}
```

```
void hanoi(int n,char one,char two,
                                char three)
{  void move(char x,char y);
   if(n==1)
      move(one,three);
   else
   {  hanoi(n-1,one,three,two);
      move(one,three);
      hanoi(n-1,two,one,three);
    }
}
```

```
void move(char x,char y)
 {
    printf("%c-->%c\n",x,y);
 }
```

```
the number of diskes:3
move 3 diskes:
A-->C
A-->B
C-->B
A-->C
B-->A
B-->C
A-->C
```

7.7数组作为函数参数

7.7.1数组元素作函数实参

例7.9 输入10个数，要求输出其中值最大的元素和该数是第几个数。

7.7.1数组元素作函数实参

- 解题思路：
  - 定义数组a，用来存放10个数
  - 设计函数max，用来求两个数中的大者
  - 在主函数中定义变量m，初值为a[0]，每次调用max函数后的返回值存放在m中
  - 用“打擂台”算法，依次将数组元素a[1]到a[9]与m比较，最后得到的m值就是10个数中的最大者

```
#include <stdio.h>
int main()
{  int max(int x,int y);
   int a[10],m,n,i;
   printf("10 integer numbers:\n");
   for(i=0;i<10;i++)
      scanf("%d",&a[i]);
   printf("\n");
```

```
10 integer numbers:
4 7 0 -3 4 34 67 -42 31 -76
```

```
    for(i=1,m=a[0],n=0;i<10;i++)
    { if (max(m,a[i])>m)
      {  m=max(m,a[i]);
         n=i;
       }
     }
     printf("largest number is %d\n",m);
     printf("%dth number.\n",n+1);
}
int max(int x,int y)
{  return(x>y?x:y);  }
```

```
largest number is 67
7th number.
```

7.7.2数组名作函数参数

- 除了可以用数组元素作为函数参数外，还可以用数组名作函数参数(包括实参和形参)
- 用数组元素作实参时，向形参变量传递的是数组元素的值
- 用数组名作函数实参时，向形参 传递的是数组首元素的地址

7.7.2数组名作函数参数

例7.10 有一个一维数组score，内放10个学生成绩，求平均成绩。

- 解题思路：
  - 用函数average求平均成绩，用数组名作为函数实参，形参也用数组名
  - 在average函数中引用各数组元素，求平均成绩并返回main函数

```
#include <stdio.h>
int main()
{ float average(float array[10]);
  float score[10],aver;  int i;
  printf("input 10 scores:\n");
  for(i=0;i<10;i++)
     scanf("%f",&score[i]);
  printf("\n");
  aver=average(score);
  printf("%5.2f\n",aver);
  return 0;
}
```

定义实参数组

定义形参数组

```
float average(float array[10])
{  int i;
   float aver,sum=array[0];
   for(i=1;i<10;i++)
      sum=sum+array[i];
   aver=sum/10;
   return(aver);
}
```

相当于score[0]

相当于score[i]

```
input 10 scores:
100 56 78 98 67.5 99 54 88.5 76 58

77.50
```

例7.11 有两个班级，分别有35名和30名学生，调用一个average函数，分别求这两个班的学生的平均成绩。

- 解题思路：
  - 需要解决怎样用同一个函数求两个不同长度的数组的平均值的问题
  - 定义**average**函数时不指定数组的长度，在形参表中增加一个整型变量**i**
  - 从主函数把数组实际长度从实参传递给形参**i**
  - 这个**i**用来在**average**函数中控制循环的次数
  - 为简化，设两个班的学生数分别为**5**和**10**

```
#include <stdio.h>
int main()
{ float average(float array[ ],int n);
  float score1[5]={98.5,97,91.5,60,55};
  float score2[10]={67.5,89.5,99,69.5,
                    77,89.5,76.5,54,60,99.5};
  printf("%6.2f\n",average(score1,5));
  printf("%6.2f\n",average(score2,10));
  return 0;
}
```

调用形式为average(score1,5)时

```
float average(float array[ ],int n)
 { int i;
   float aver,sum=array[0];
   for(i=1;i<n;i++)
      sum=sum+array[i];
   aver=sum/n;
   return(aver);
}
```

相当于5

相当于score1[0]

相当于score1[i]

调用形式为average(score2,10)时

float array[]=score2

```
float average(float array[ ],int n)
 { int i;
   float aver,sum=array[0];
   for(i=1;i<n;i++)
      sum=sum+array[i];
   aver=sum/n;
   return(aver);
}
```

相当于10

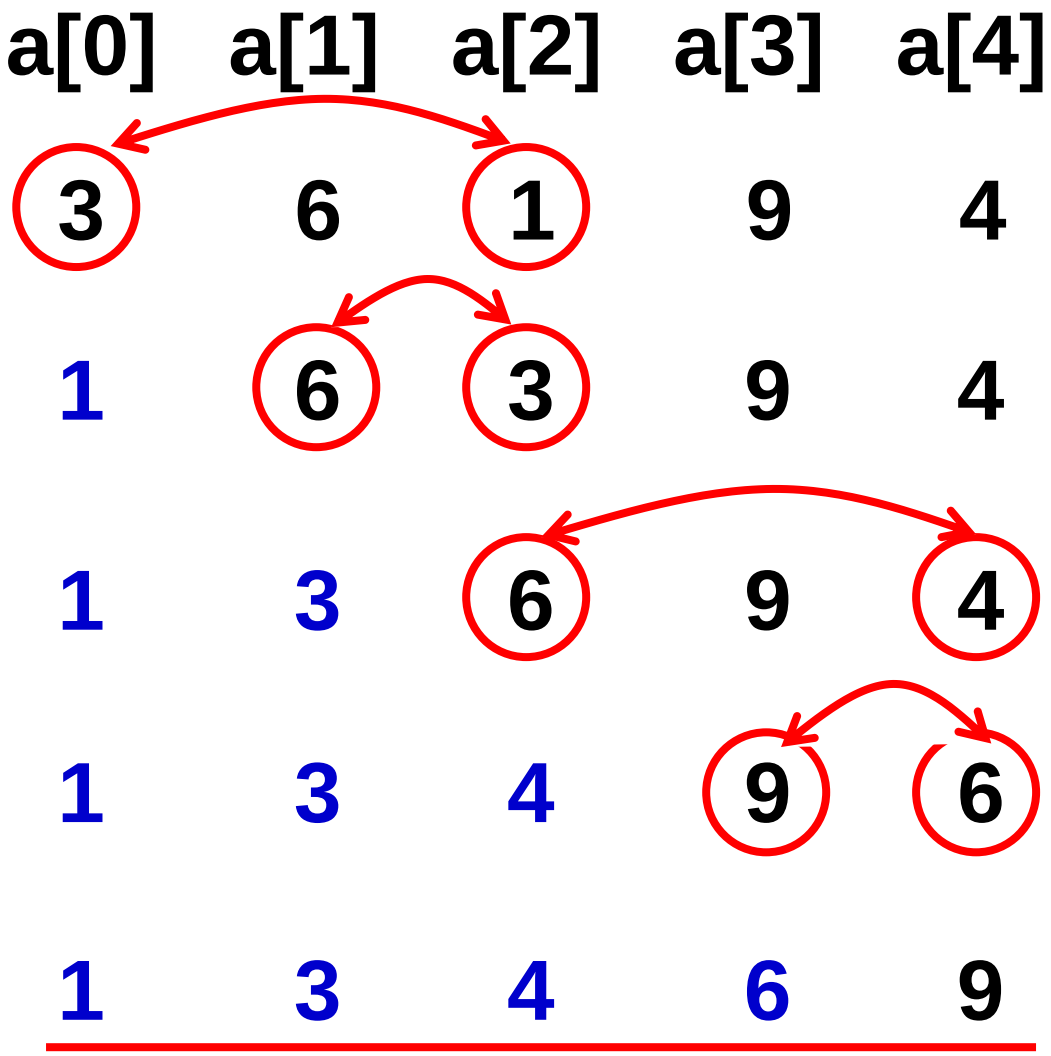
相当于score2[0]

相当于score2[i]

```
80.40
78.20
```

例7.12用选择法对数组中10个整数按由小到大排序。

- 解题思路：
  - 所谓选择法就是先将10个数中最小的数与a[0]对换；再将a[1]到a[9]中最小的数与a[1]对换……每比较一轮，找出一个未经排序的数中最小的一个
  - 共比较9轮



小到大排序

```
#include <stdio.h>
int main()
{ void sort(int array[],int n);
  int a[10],i;
  printf("enter array:\n");
  for(i=0;i<10;i++)  scanf("%d",&a[i]);
  sort(a,10);
  printf("The sorted array:\n");
  for(i=0;i<10;i++)   printf("%d ",a[i]);
  printf("\n");
  return 0;
}
```

```
void sort(int array[],int n)
 { int i,j,k,t;
   for(i=0;i<n-1;i++)
   { k=i;
     for(j=i+1;j<n;j++)
       if(array[j]<array[k])    k=j;
     t=array[k];
     array[k]=array[i];
     array[i
   }
 }
```

在sort[i]~sort[9]中，最小数与sort[i]对换

```
enter array:
45 2 9 0 -3 54 12 5 66 33
The sorted array:
-3 0 2 5 9 12 33 45 54 66
```

7.7.3多维数组名作函数参数

例**7.13** 有一个 3 × 4 的矩阵，求所有元素中的最大值。

- 解题思路：先使变量**max**的初值等于矩阵中第一个元素的值，然后将矩阵中各个元素的值与**max**相比，每次比较后都把“大者”存放在**max**中，全部元素比较完后，**max** 的值就是所有元素的最大值。

不能省略
要与形参数组第二维大小相同

```
#include <stdio.
int main()
{ int max_value(int array[][4]);
  int a[3][4]={{1,3,5,7},{2,4,6,8},
                         {15,17,34,12}};
  printf("Max value is %d\n",
                          max_value(a));
  return 0;
}
```

要与实参数组第二维大小相同

```
int max_value(int array[][4])
{  int i,j,max;
   max = array[0][0];
   for (i=0;i<3;i++)
      for(j=0;j<4;j++)
         if (array[i][j]>max)
            max = array[i][j];
   return (max);
}
```

7.8局部变量和全局变量

7.8.1 局部变量

7.8.2 全局变量

7.8.1 局部变量

- 定义变量可能有三种情况：
  - 在函数的开头定义
  - 在函数内的复合语句内定义
  - 在函数的外部定义

7.8.1 局部变量

- 在一个函数内部定义的变量只在本函数范围内有效
- 在复合语句内定义的变量只在本复合语句范围内有效
- 在函数内部或复合语句内部定义的变量称为“局部变量”

```
float f1( int a)
{ int b,c;
   ......
}
```

a、b、c仅在此函数内有效

```
char f2(int x,int y)
{ int i,j;
   ......
}
```

x、y、i、j仅在此函数内有效

```
int main( )
{ int m,n;
   ......
   return 0;
}
```

m、n仅在此函数内有效

```
float f1( int a)
{ int b,c;
   ......
}
char f2(int x,i
{ int i,j;
   ......
}
int main( )
{ int a,b;
   ......
   return 0;
}
```

类似于不同班同名学生

a、b也仅在此函数内有效

```
int main ( )
{ int a,b,c;
    ......
  { int c;
    c=a+b;
    ......
  }
   ......
}
```

a、b仅在此复合语句内有效

c仅在此复合语句内有效

7.8.2全局变量

- 在函数内定义的变量是局部变量，而在函数之外定义的变量称为外部变量
- 外部变量是全局变量(也称全程变量)
- 全局变量可以为本文件中其他函数所共用
- 有效范围为从定义变量的位置开始到本源文件结束

```
int p=1,q=5
float f1(int a)
{  int b,c;   ……  }
char c1,c2;
char f2 (int x, int y)
{  int i,j;   ……  }
int main ( )
{  int m,n;
   ……
   return 0;
}
```

p、q、c1、c2
为全局变量

```
#include<stdio>
int p=1,q=5
float f1(int a)
{  int b,c;   ……  }
char c1,c2;
char f2 (int x, int y)
{  int i,j;   ……  }
int main ( )
{  int m,n;
   ……
   return 0;
}
```

p、q的有效范围

c1、c2的有效范围

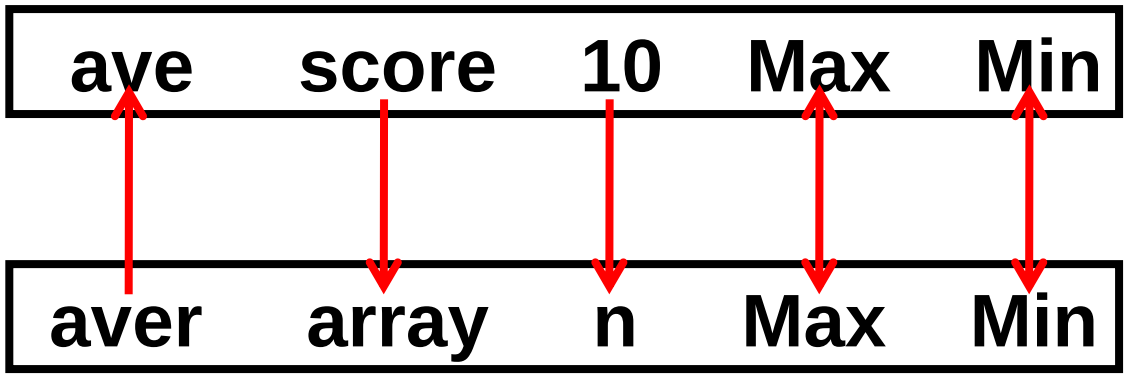
例7.14 有一个一维数组，内放10个学生成绩，写一个函数，当主函数调用此函数后，能求出平均分、最高分和最低分。

- 解题思路：调用一个函数可以得到一个函数返回值，现在希望通过函数调用能得到3个结果。可以利用全局变量来达到此目的。

```
#include <stdio.h>
float Max=0,Min=0;
int main()
{ float average(float array[ ],int n,float
  mm[]);
  float ave,score[10],MM[2];  int i;
  printf("Please enter 10 scores:\n");
  for(i=0;i<10;i++)
     scanf("%f",&score[i]);
  ave=average(score,10, MM);
  printf("max=%6.2f\nmin=%6.2f\n
       average=%6.2f\n",Max,Min,ave);
  return 0;
 }
```

```
float average(float array[ ],int n, float
   mm[])
{ int i;  float aver,sum=array[0];
   mm[0] = mm[1] = array[0];
   Max=Min=array[0];
   for(i=1;i<n;i++)
   { if(array[i]>Max) Max=array[i];
     else if(array[i]<Min) Min=array[i];
     sum=sur
    }
    aver=sum
    return(ave
```

```
Please enter 10 scores:
89 95 87.5 100 67.5 97 59 84 73 90
max=100.00
min= 59.00
average= 84.20
```



main
函数

average
函数

建议不在必要时不要使用全局变量

例7.15 若外部变量与局部变量同名，分析结果。

```
#include <stdio.h>
int a=3,b=5;
int main()
{ int max(int a,int b);
  int a=8;
  printf("max=%d\n",max(a,b));
  return 0;
}
int max(int a,int b)
{ int c;
  c=a>b?a:b;
  return(c);
}
```

b为全部变量

a为局部变量，仅在此函数内有效

```
#include <stdio.h>
int a=3,b=5;
int main()
{ int max(int a,int b);
  int a=8;
  printf("max=%d\n",max(a,b));
  return 0;
}
int max(int a,int b)
{ int c;
  c=a>b?a:b;
  return(c);
}
```

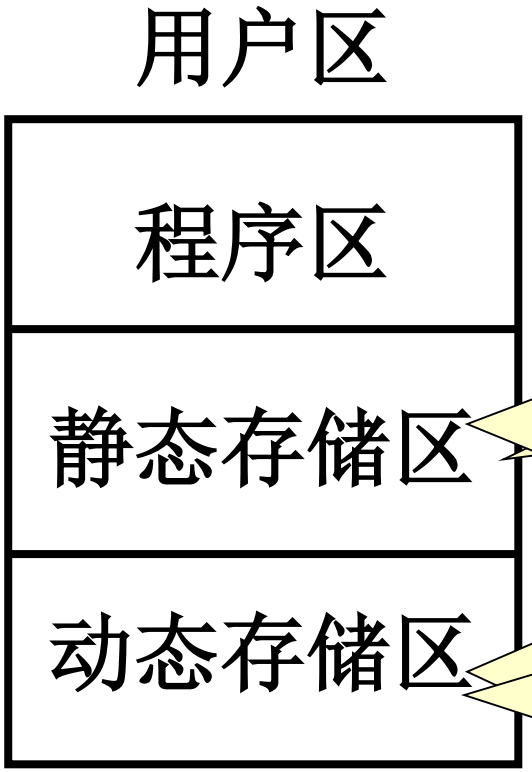
```
max=8
```

a、b为局部变量，仅在此函数内有效

7.9变量的存储方式和生存期

7.9.1动态存储方式与静态存储方式

- 从变量的作用域的角度来观察，变量可以分为全局变量和局部变量
- 从变量值存在的时间(即生存期)观察，变量的存储有两种不同的方式：静态存储方式和动态存储方式
  - ◆静态存储方式是指在程序运行期间由系统分配固定的存储空间的方式
  - ◆动态存储方式是在程序运行期间根据需要进行动态的分配存储空间的方式



程序开始执行时给全局变量分配存储区，程序执行完毕就释放。在程

函数调用开始时分配，函数结束时释放。在程序执行过程中，这种分配和释放是动态的

- 每一个变量和函数都有两个属性：数据类型和数据的存储类别
  - 数据类型，如整型、浮点型等
  - 存储类别指的是数据在内存中存储的方式(如静态存储和动态存储)
  - 存储类别包括：

    自动的、静态的、寄存器的、外部的
  - 根据变量的存储类别，可以知道变量的作用域和生存期

7.9.2 局部变量的存储类别

1.自动变量(auto变量)

- ◆局部变量，如果不专门声明存储类别，都是动态地分配存储空间的
- ◆调用函数时，系统会给局部变量分配存储空间，调用结束时就自动释放空间。因此这类局部变量称为自动变量
- ◆自动变量用关键字**auto**作存储类别的声明

7.9.2 局部变量的存储类别

```
int f(int a)
{
    auto int b,c=3;
}
```

可以省略

7.9.2 局部变量的存储类别

2.静态局部变量(static局部变量)

- 希望函数中的局部变量在函数调用结束后不消失而继续保留原值，即其占用的存储单元不释放，在下一次再调用该函数时，该变量已有值(就是上一次函数调用结束时的值)，这时就应该指定该局部变量为“静态局部变量”，用关键字**static**进行声明

例7.16 考察静态局部变量的值。

```
#include <stdio.h>
int main()
{ int f(int);
  int a=2,i;
  for(i=0;i<3;i++)
    printf("%d\n",f(a));
  return 0;
}
```

```
int f(int a)
{ auto  int b=0;
  static int c=3;
  b=b+1;
  c=c+1;
  return(a+b+c);
}
```

调用三次

每调用一次，开辟新a和b，但c不是

例7.16 考察静态局部变量的值。

```
#include <stdio.h>
int main()
{ int f(int);
  int a=2,i;
  for(i=0;i<3;i++)
    printf("%d\n",f(a));
  return 0;
}
```

```
int f(int a)
{ auto  int b=0;
  static int c=3;
  b=b+1;
  c=c+1;
  return(a+b+c);
}
```

b	c
0	3

第一次调用开始

例7.16 考察静态局部变量的值。

```
#include <stdio.h>
int main()
{ int f(int);
  int a=2,i;
  for(i=0;i<3;i++)
    printf("%d\n",f(a));
  return 0;
}
```

```
int f(int a)
{ auto  int b=0;
  static int c=3;
  b=b+1;
  c=c+1;
  return(a+b+c);
}
```

b	c
1	4

第一次调用期间

例7.16 考察静态局部变量的值。

```
#include <stdio.h>
int main()
{ int f(int);
  int a=2,i;
  for(i=0;i<3;i++)
    printf("%d\n",f(a));
  return 0;
}
```

```
int f(int a)
{ auto  int b=0;
  static int c=3;
  b=b+1;
  c=c+1;
  return(a+b+c);
}
```

7

c

4

第一次调用结束

例7.16 考察静态局部变量的值。

```
#include <stdio.h>
int main()
{ int f(int);
  int a=2,i;
  for(i=0;i<3;i++)
    printf("%d\n",f(a));
  return 0;
}
```

```
int f(int a)
{ auto  int b=0;
  static int c=3;
  b=b+1;
  c=c+1;
  return(a+b+c);
}
```

b	c
0	4

第二次调用开始

例7.16 考察静态局部变量的值。

```
#include <stdio.h>
int main()
{ int f(int);
  int a=2,i;
  for(i=0;i<3;i++)
    printf("%d\n",f(a));
  return 0;
}
```

```
int f(int a)
{ auto  int b=0;
  static int c=3;
  b=b+1;
  c=c+1;
  return(a+b+c);
}
```

b	c
1	5

第二次调用期间

例7.16 考察静态局部变量的值。

```
#include <stdio.h>
int main()
{ int f(int);
  int a=2,i;
  for(i=0;i<3;i++)
    printf("%d\n",f(a));
  return 0;
}
```

```
int f(int a)
{ auto  int b=0;
  static int c=3;
  b=b+1;
  c=c+1;
  return(a+b+c);
}
```

8

c

5

第二次调用结束

例7.16 考察静态局部变量的值。

```
#include <stdio.h>
int main()
{ int f(int);
  int a=2,i;
  for(i=0;i<3;i++)
    printf("%d\n",f(a));
  return 0;
}
```

```
int f(int a)
{ auto  int b=0;
  static int c=3;
  b=b+1;
  c=c+1;
  return(a+b+c);
}
```

b	c
0	5

第三次调用开始

例7.16 考察静态局部变量的值。

```
#include <stdio.h>
int main()
{ int f(int);
  int a=2,i;
  for(i=0;i<3;i++)
    printf("%d\n",f(a));
  return 0;
}
```

```
int f(int a)
{ auto  int b=0;
  static int c=3;
  b=b+1;
  c=c+1;
  return(a+b+c);
}
```

b	c
1	6

第三次调用期间

例7.16 考察静态局部变量的值。

```
#include <stdio.h>
int main()
{ int f(int);
  int a=2,i;
  for(i=0;i<3;i++)
    printf("%d\n",f(a));
  return 0;
}
```

```
int f(int a)
{ auto  int b=0;
  static int c=3;
  b=b+1;
  c=c+1;
  return(a+b+c);
}
```

9

c

6

第三次调用结束

例7.16 考察静态局部变量的值。

```
#include <stdio.h>
int main()
{ int f(int);
  int a=2,i;
  for(i=0;i<3;i++)
    printf("%d\n",f(a));
  return 0;
}
```

```
int f(int a)
{ auto  int b=0;
  static int c=3;
  b=b+1;
  c=c+1;
  return(a+b+c);
}
```

整个程序结束

```
7
8
9
```

例7.16 考察静态局部变

```
#include <stdio.h>
int main()
{ int f(int);
  int a=2,i;
  for(i=0;i<3;i++)
    printf("%d\n",f(a));
  return 0;
}
```

```
int f(int a)
{ auto  int b=0;
  static int c=3;
  b=b+1;
  c=c+1;
}
```

在函数调用时赋初值

在编译时赋初值

例7.16 考察静态局部变

```
#include <stdio.h>
int main()
{ int f(int);
  int a=2,i;
  for(i=0;i<3;i++)
    printf("%d\n",f(a));
  return 0;
}
```

```
int f(int a)
{ auto  int b=0;
  static int c=3;
  b=b+1;
  c=c+1;
}
```

若不赋初值，不确定

若不赋初值，是0

例7.16 考察静态局部变量的值。

```
#include <stdio.h>
int main()
{ int f(int);
  int a=2,i;
  for(i=0;i<3;i++)
    printf("%d\n",f(a));
  return 0;
}
```

```
int f(int a)
{ auto  int b=0;
  static int c=3;
  b=b+1;
  c=c+1;
}
```

仅在本函数内有效

例7.17 输出1到5的阶乘值。

- 解题思路：可以编一个函数用来进行连乘，如第1次调用时进行1乘1，第2次调用时再乘以2，第3次调用时再乘以3，依此规律进行下去。

```
#include <stdio.h>
int main()
{ int fac(int n);
   int i;
   for(i=1;i<=5;i++)
       printf("%d!=%d\n",i,fac(i));
   return 0;
}
int fac(int n)
{ static int f=1;
   f=f*n;
   return(f);
}
```

若非必要，不要多用静态局部变量

```
1!=1
2!=2
3!=6
4!=24
5!=120
```

3. 寄存器变量(register变量)

- 一般情况下，变量（包括静态存储方式和动态存储方式）的值是存放在内存中的
- 寄存器变量允许将局部变量的值放在CPU中的寄存器中
- 现在的计算机能够识别使用频繁的变量，从而自动地将这些变量放在寄存器中，而不需要程序设计者指定

7.9.3 全局变量的存储类别

- 全局变量都是存放在静态存储区中的。因此它们的生存期是固定的，存在于程序的整个运行过程
- 一般来说，外部变量是在函数的外部定义的全局变量，它的作用域是从变量的定义处开始，到本程序文件的末尾。在此作用域内，全局变量可以为程序中各个函数所引用。

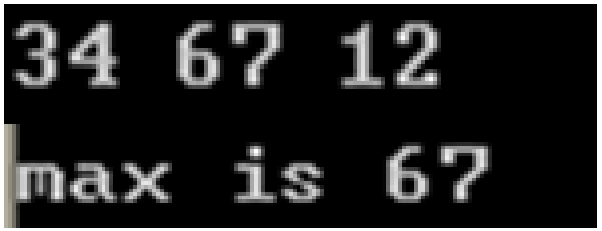
1. 在一个文件内扩展外部变量的作用域

- 外部变量有效的作用范围只限于定义处到本文件结束。
- 如果用关键字**extern**对某变量作“外部变量声明”，则可以从“声明”处起，合法地使用该外部变量

例7.18 调用函数，求3个整数中的大者。

- 解题思路：用extern声明外部变量，扩展外部变量在程序文件中的作用域。

```
#include <stdio.h>
int main()
{ int max( );
   extern int A,B,C;
   scanf("%d %d %d",&A,&B,&C);
   printf("max is %d\n",max());
   return 0;
}
int A ,B ,C;
int max( )
{ int m;
   m=A>B?A:B;
   if (C>m) m=C;
   return(m);
}
```



2. 将外部变量的作用域扩展到其他文件

- 如果一个程序包含两个文件，在两个文件中都要用到同一个外部变量**Num**，不能分别在两个文件中各自定义一个外部变量**Num**
- 应在任一个文件中定义外部变量**Num**，而在另一文件中用**extern**对**Num**作“外部变量声明”
- 在编译和连接时，系统会由此知道**Num**有“外部链接”，可以从别处找到已定义的外部变量**Num**，并将在另一文件中定义的外部变量**num**的作用域扩展到本文件

例7.19 给定b的值，输入a和m，求a*b和a^m的值。

- 解题思路：
  - 分别编写两个文件模块，其中文件file1包含主函数，另一个文件file2包含求a^m的函数。
  - 在file1文件中定义外部变量A，在file2中用extern声明外部变量A，把A的作用域扩展到file2文件。

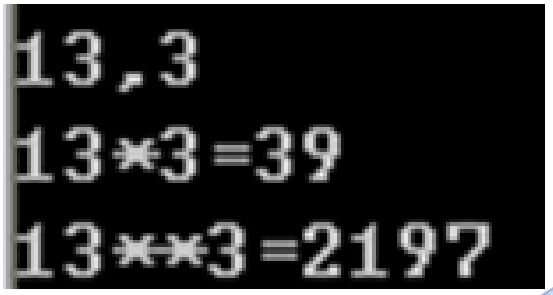
文件**file1.c**：

```
#include <stdio.h>
int A;
int main()
{ int power(int);
  int b=3,c,d,m;
  scanf("%d,%d",&A,&m);
  c=A*b;
  printf("%d*%d=%d\n",A,b,c);
  d=power(m);
  printf("%d**%d=%d\n",A,m,d);
  return 0;
}
```

编译和运行包括多个文件的程序，可参考《C程序设计学习辅导》一书的“C语言上机指南”部分

文件file2.c：

```
extern  A;
int power(int n)
{ int i,y=1;
   for(i=1;i<=n;i++)
      y*=A;
   return(y);
}
```



3.将外部变量的作用域限制在本文件中

- 有时在程序设计中希望某些外部变量只限于被本文件引用。这时可以在定义外部变量时加一个static声

只能用于本文件

本文件仍然不能用

file1.c

```
static int A;
int main ( )
{
    ……
}
```

file2.c

```
extern   A;
void fun (int n)
{  ……
   A=A*n;
   ……
}
```

- 说明:
  - 不要误认为对外部变量加**static**声明后才采取静态存储方式，而不加**static**的是采取动态存储
  - 声明局部变量的存储类型和声明全局变量的存储类型的含义是不同的
  - 对于局部变量来说，声明存储类型的作用是指定变量存储的区域以及由此产生的生存期的问题，而对于全局变量来说，声明存储类型的作用是变量作用域的扩展问题

- 用**static** 声明一个变量的作用是：

  (1) 对局部变量用**static**声明，把它分配在静态存储区，该变量在整个程序执行期间不释放，其所分配的空间始终存在。

  (2) 对全局变量用**static**声明，则该变量的作用域只限于本文件模块**(**即被声明的文件中**)**。

- 注意：用**auto**、**register**、**static**声明变量时，是在定义变量的基础上加上这些关键字，而不能单独使用。
- 下面用法不对：

```
int a;
static a;
```

编译时会被认为“重新定义”。

7.9.4 存储类别小结

➢对一个数[illegible]两种属性：

◆数据类型[illegible]两个关键字

例如：

static int a;

auto char c;

register int d[illegible]

◆可以用**extern**声[illegible]已定义的外部变量

例如： **extern b;**

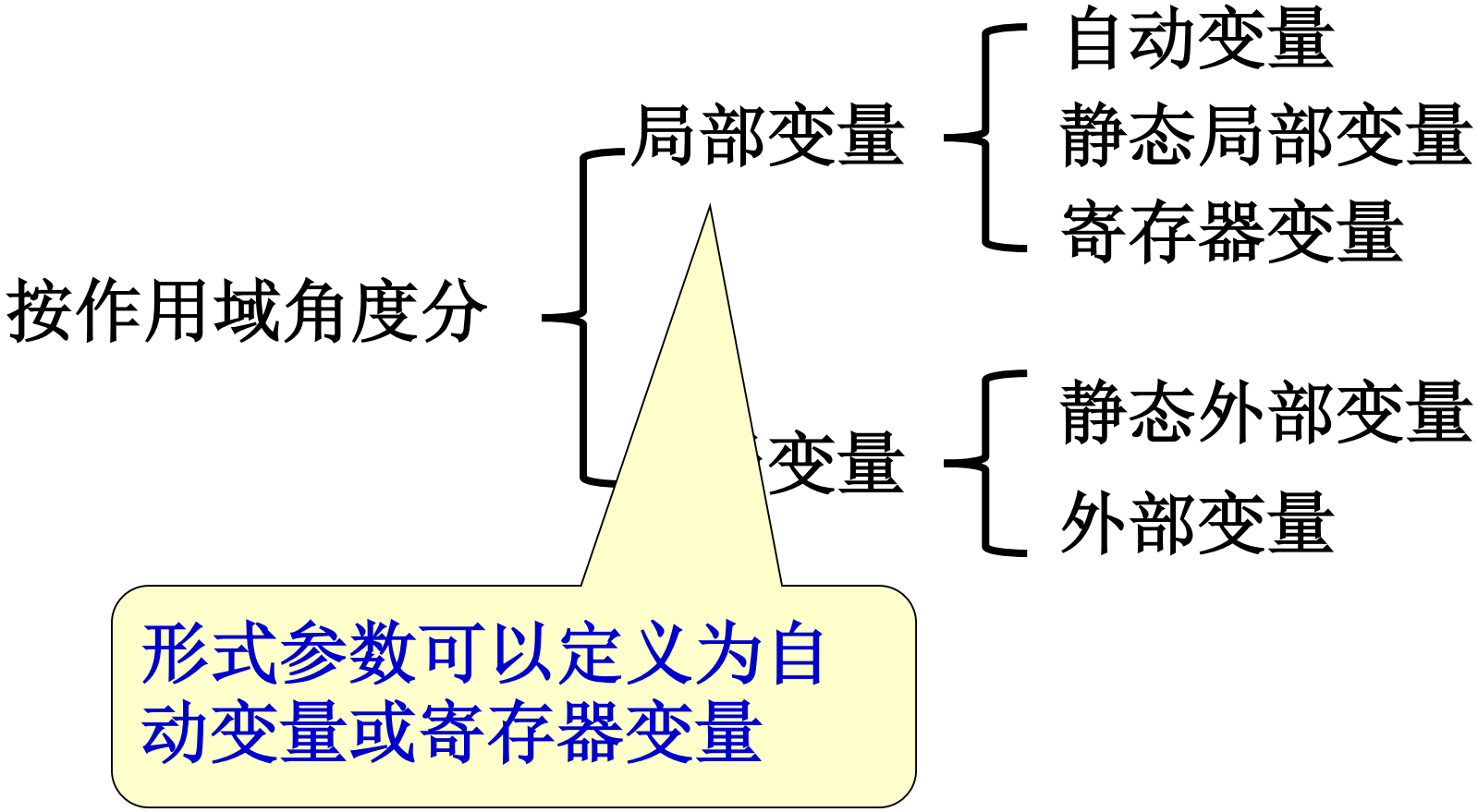
静态局部整型变量或
静态外部整型变量

自动变量，在

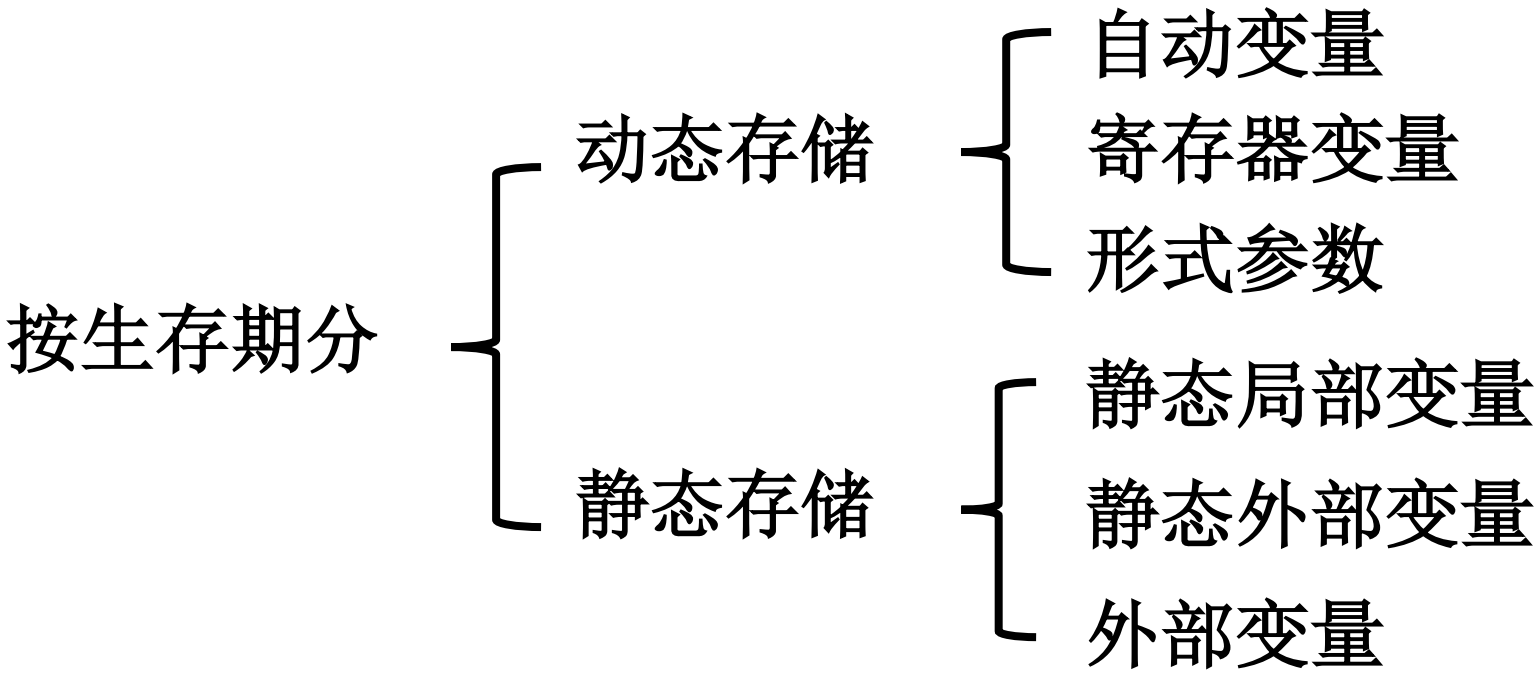
寄存器变量，
在函数内定义

将已定义的外部变量
b的作用域扩展至此

（1）从作用域角度分，有局部变量和全局变量。它们采用的存储类别如下：



（2）从变量存在的时间区分,有动态存储和静态存储两种类型。静态存储是程序整个运行时间都存在,而动态存储则是在调用函数时临时分配单元



（3）从变量值存放的位置来区分,可分为:

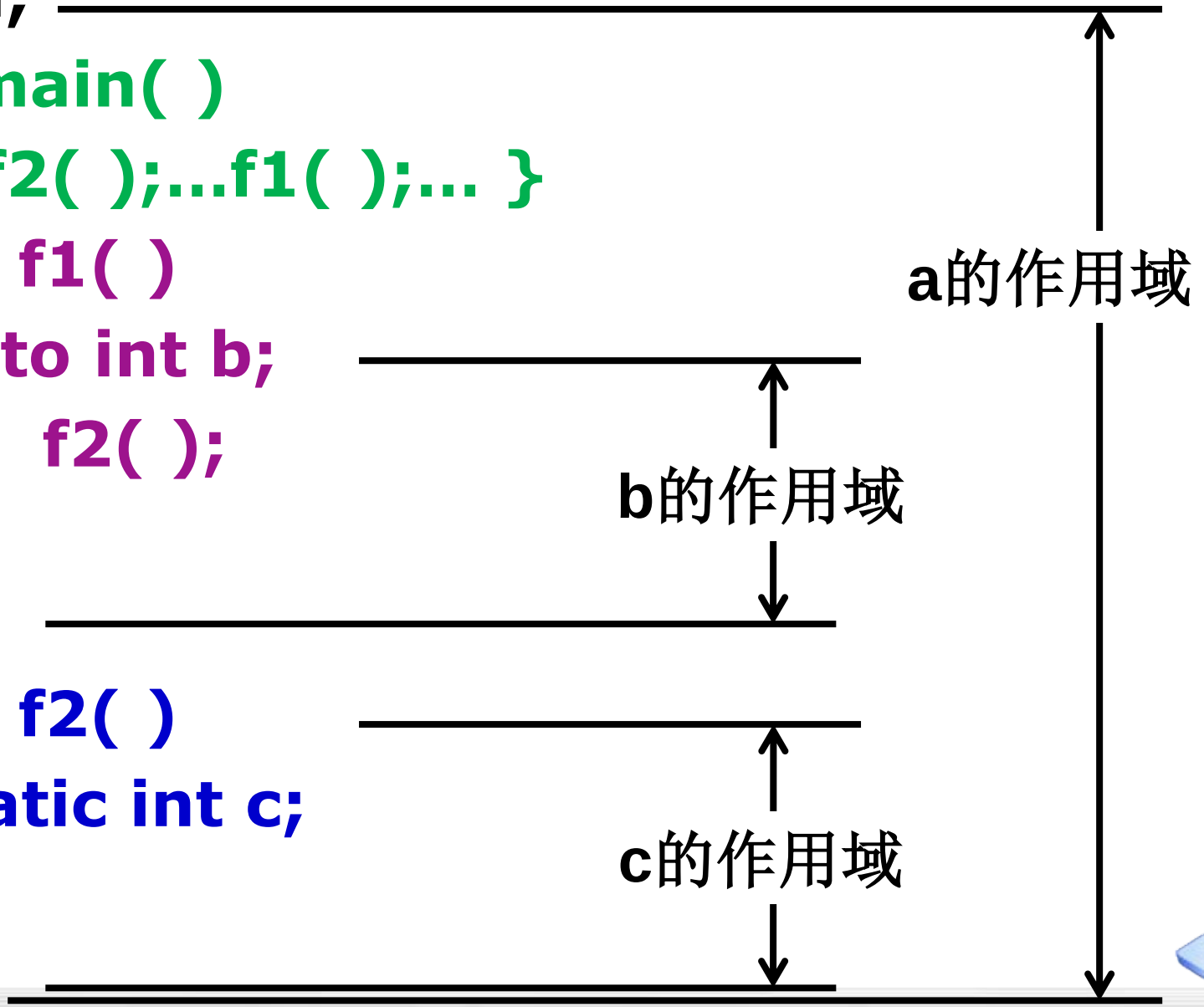
- 按变量值存放的位置分
  - 内存中静态存储区
    - 静态局部变量
    - 静态外部变量
    - 外部变量
  - 内存中动态存储区——自动变量和形式参数
  - CPU中的寄存器 —— 寄存器变量

（4）关于作用域和生存期的概念

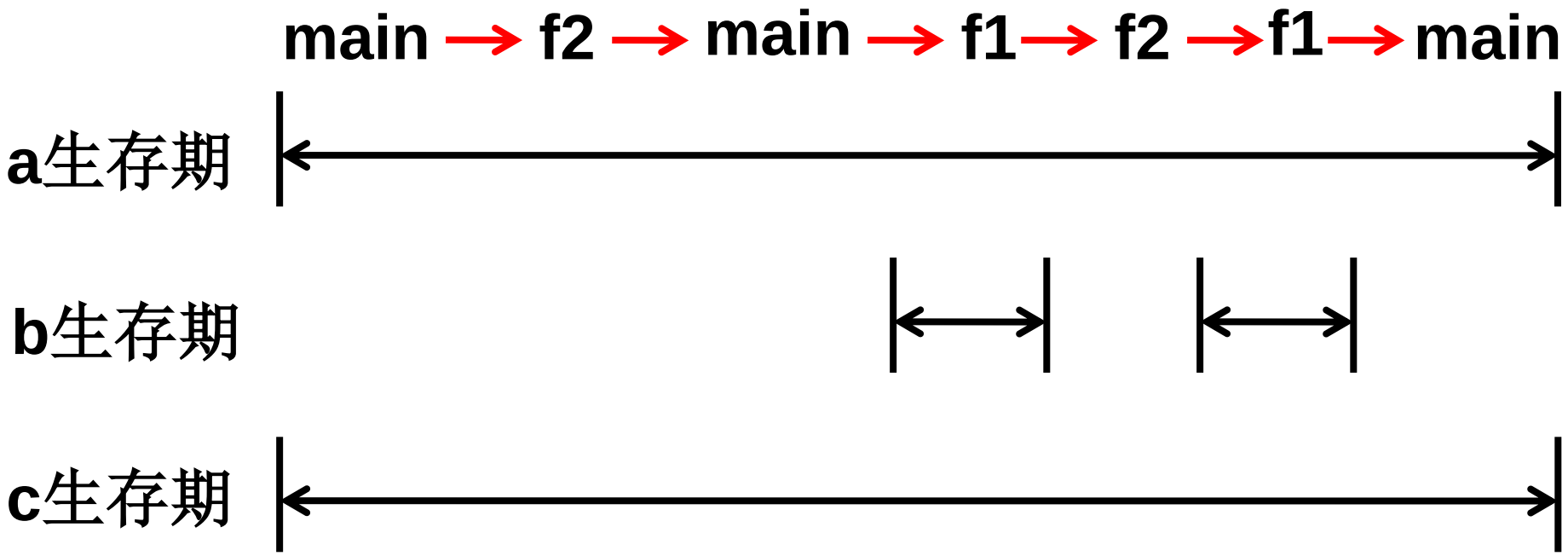
- 对一个变量的属性可以从两个方面分析：
  - 作用域：如果一个变量在某个文件或函数范围内是有效的，就称该范围为该变量的作用域
  - 生存期：如果一个变量值在某一时刻是存在的，则认为这一时刻属于该变量的生存期
- 作用域是从空间的角度，生存期是从时间的角度
- 二者有联系但不是同一回事

文件file1.c

```
int a;
int main( )
{ …f2( );…f1( );… }
void f1( )
{ auto int b;
   …   f2( );
   …
}
void f2( )
{ static int c;
……
}
```



程序执行过程



各种类型变量的作用域和存在性的情况

变量存储类别	函 数 内		函 数 外	
	作用域	存在性	作用域	存在性
自动变量和寄存器变量	√	√	×	×
静态局部变量	√	√	×	√
静态外部变量	√	√	√(只限本文件)	√
外部变量	√	√	√	√

(5) static对局部变量和全局变量的作用不同

- 局部变量使变量由动态存储方式改变为静态存储方式
- 全局变量使变量局部化(局部于本文件)，但仍为静态存储方式
- 从作用域角度看，凡有static声明的，其作用域都是局限的，或者是局限于本函数内(静态局部变量)，或者局限于本文件内(静态外部变量)

7.10 关于变量的声明和定义

- 一般为了叙述方便，把建立存储空间的变量声明称定义，而把不需要建立存储空间的声明称为声明
- 在函数中出现的对变量的声明(除了用**extern**声明的以外)都是定义
- 在函数中对其他函数的声明不是函数的定义

7.11 内部函数和外部函数

7.11.1 内部函数

7.11.2 外部函数

7.11.1 内部函数

- 如果一个函数只能被本文件中其他函数所调用，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**，即：

static 类型名 函数名(形参表)

7.11.1 内部函数

- 内部函数又称静态函数，因为它是用**static**声明的
- 通常把只能由本文件使用的函数和外部变量放在文件的开头，前面都冠以**static**使之局部化，其他文件不能引用
- 提高了程序的可靠性

7.11.2 外部函数

- 如果在定义函数时，在函数首部的最左端加关键字**extern**，则此函数是外部函数，可供其他文件调用。
- 如函数首部可以为

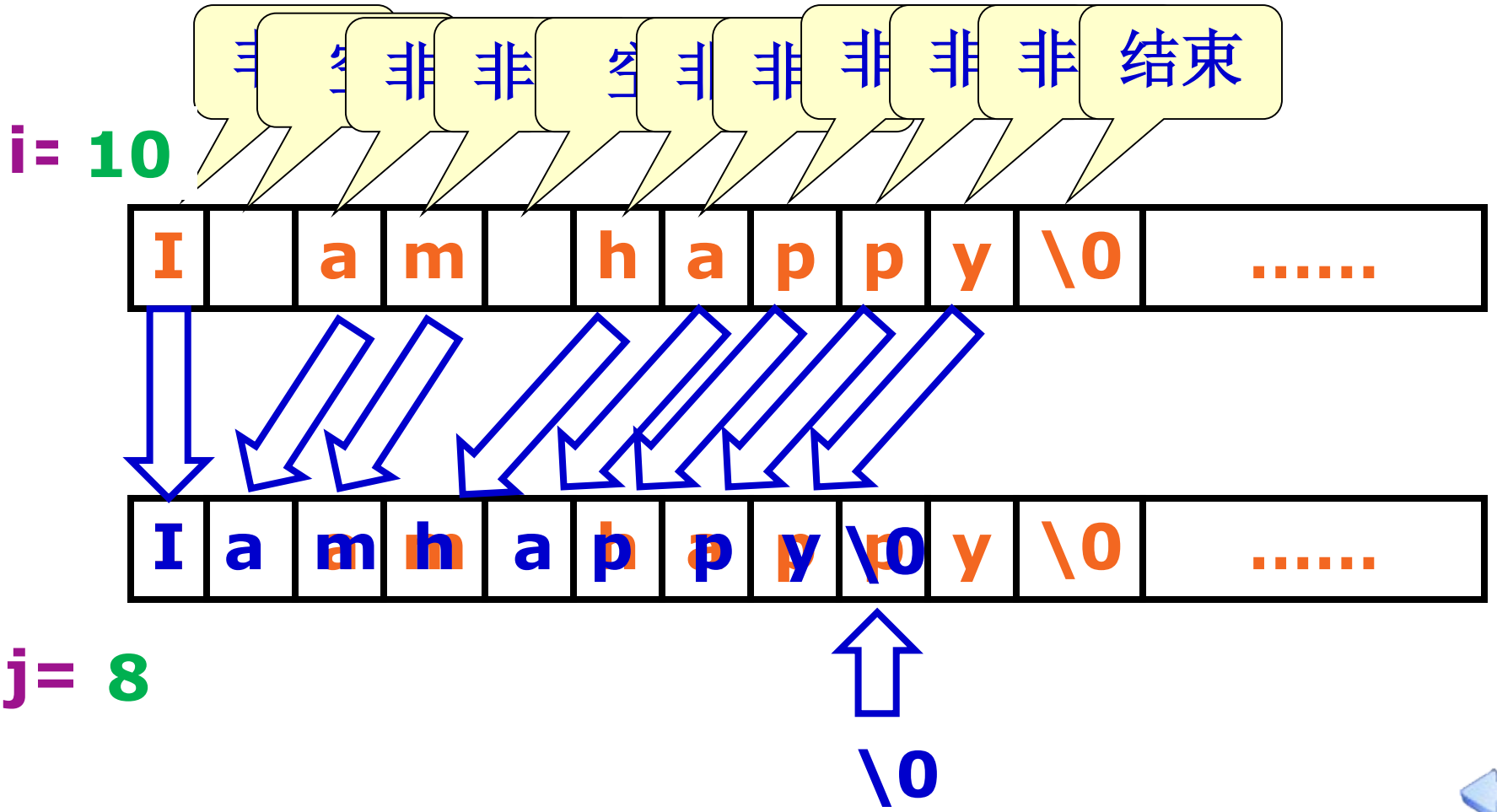
  extern int fun (int a, int b)

- 如果在定义函数时省略**extern**，则默认为外部函数

例7.20 有一个字符串，内有若干个字符，今输入一个字符，要求程序将字符串中该字符删去。用外部函数实现。

- 解题思路：
  - 分别定义3个函数用来输入字符串、删除字符、输出字符串
  - 按题目要求把以上3个函数分别放在3个文件中。main函数在另一文件中，main函数调用以上3个函数，实现题目的要求

删除空格的思路



file1（文件1）

```
#include <stdio.h>
int main()
{ extern void enter_string(char str[]);
  extern void delete_string(char str[],
                            char ch);
  extern void print_string(char str[]);
  char c,str[80];
  enter_string(str);
  scanf("%c",&c);
  delete_string(str,c);
  print_string(str);
  return 0;
}
```

声明在本函数中将要调用的已在其他文件中定义的3个函数

```
void enter_string(char str[80])
{  gets(str);  }
void delete_string(char str[],char ch)
{ int i,j;
  for(i=j=0;str[i]!='\0';i++)
    if(str[i]!=ch)  str[j++]=str[i];
  str[j]='\0';
}
void print_string(char str[])
{ printf("%s\n",str); }
```

file2（文件2）

file3（文件3）

file4（文件4）