《计算概论A》课程 程序设计部分

函数的递归调用（1）

李　戈

北京大学 信息科学技术学院 软件研究所

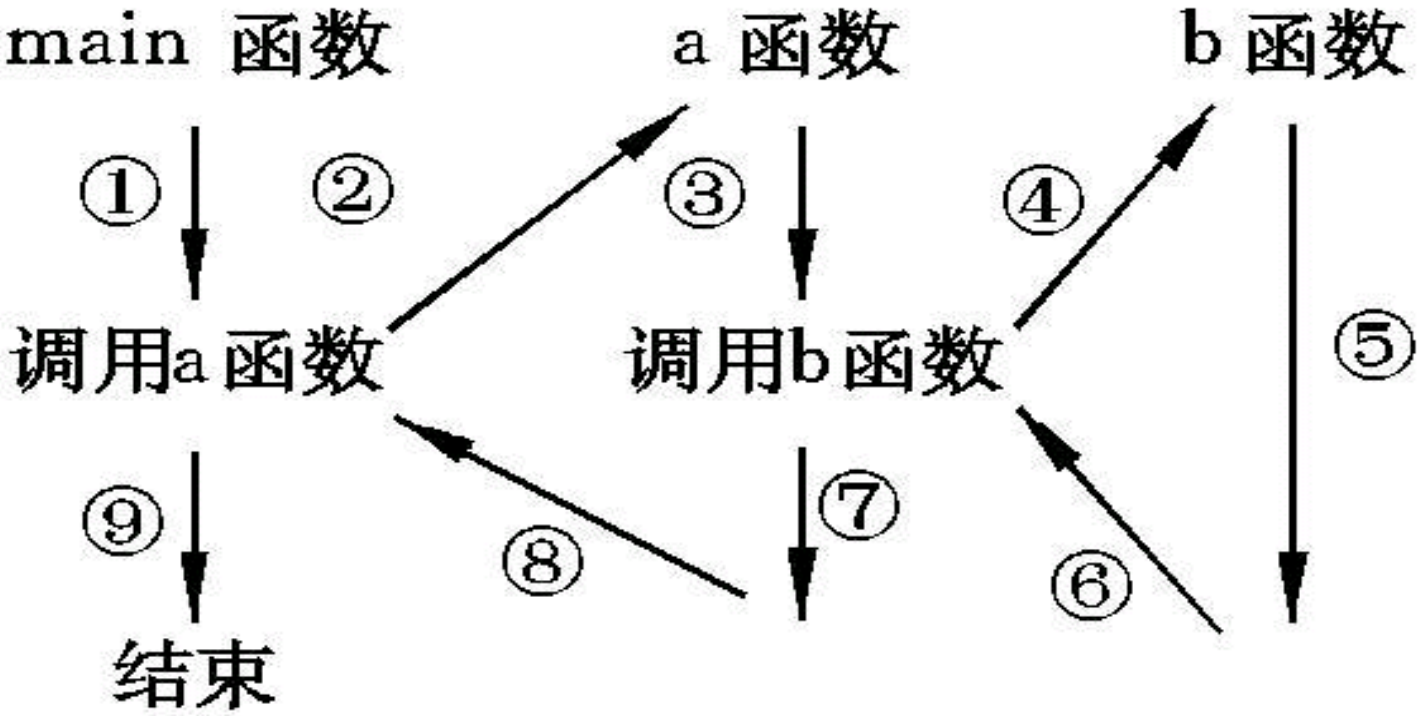
2010年11月30日

北京大学

从 嵌套 到 递归

函数的嵌套调用

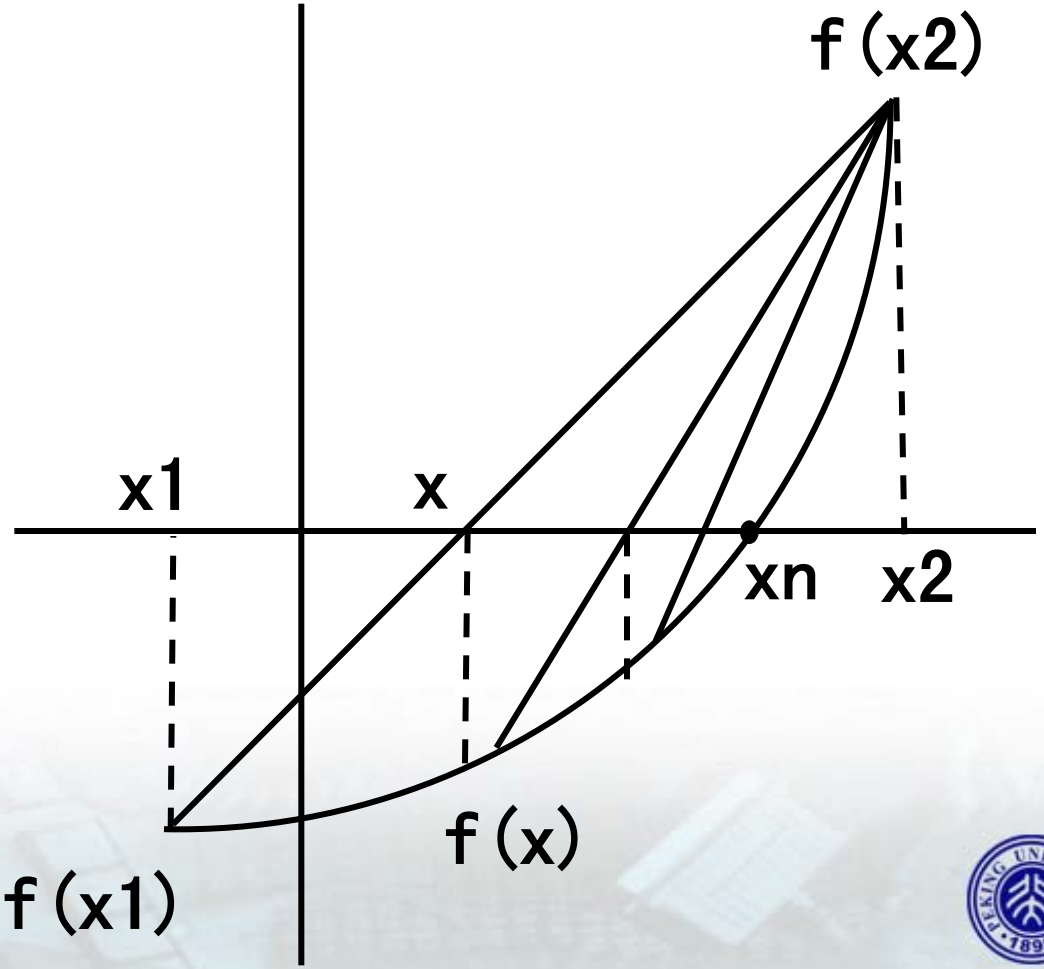
- 函数不能嵌套定义，但可以嵌套调用
  - 在调用一个函数的过程中，又调用另一函数



函数嵌套调用程序实例

[例]用弦截法求方程的根

$x^3+5x^2+16x-80=0$



北京大学

函数嵌套调用程序实例

输入x1, x2, 求f(x1), f(x2)

直到f(x1)*f(x2)<0

求f(x1)与f(x2)的连线与x轴的交点x

y=f(x), y2=f(x2)

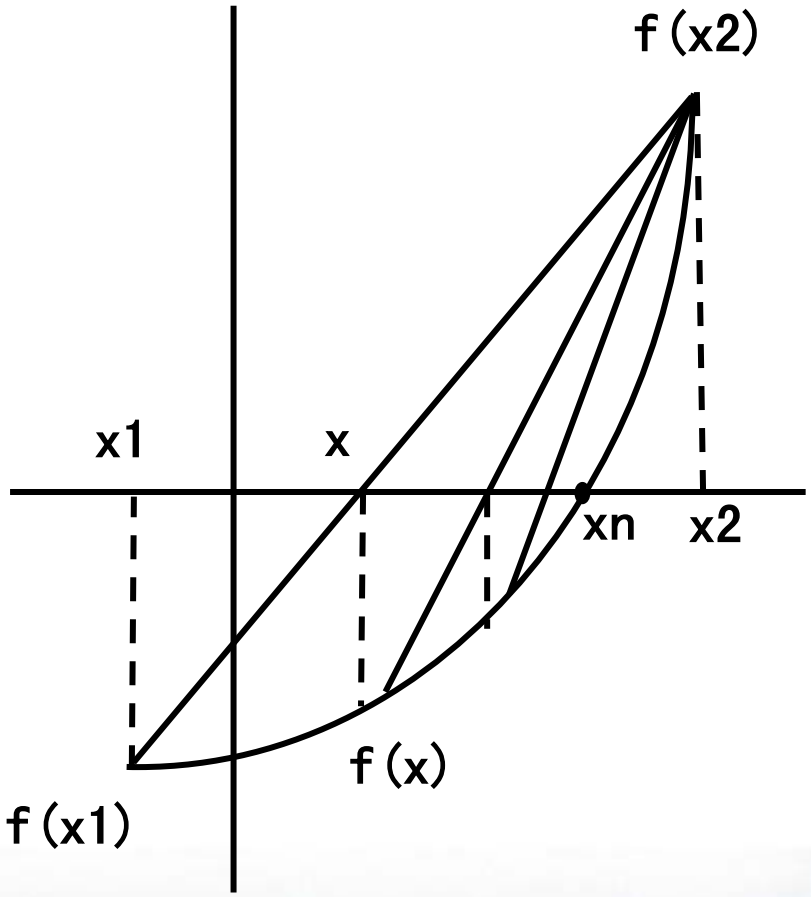
y与y2异号吗?

y　　n

x1=x　　x2=x

直到|y|<ε

root=x, 输出x

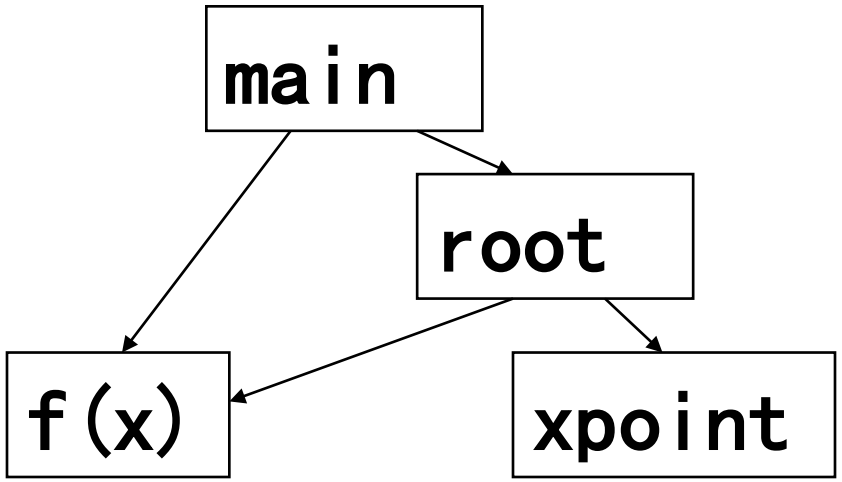


北京大学

函数嵌套调用程序实例

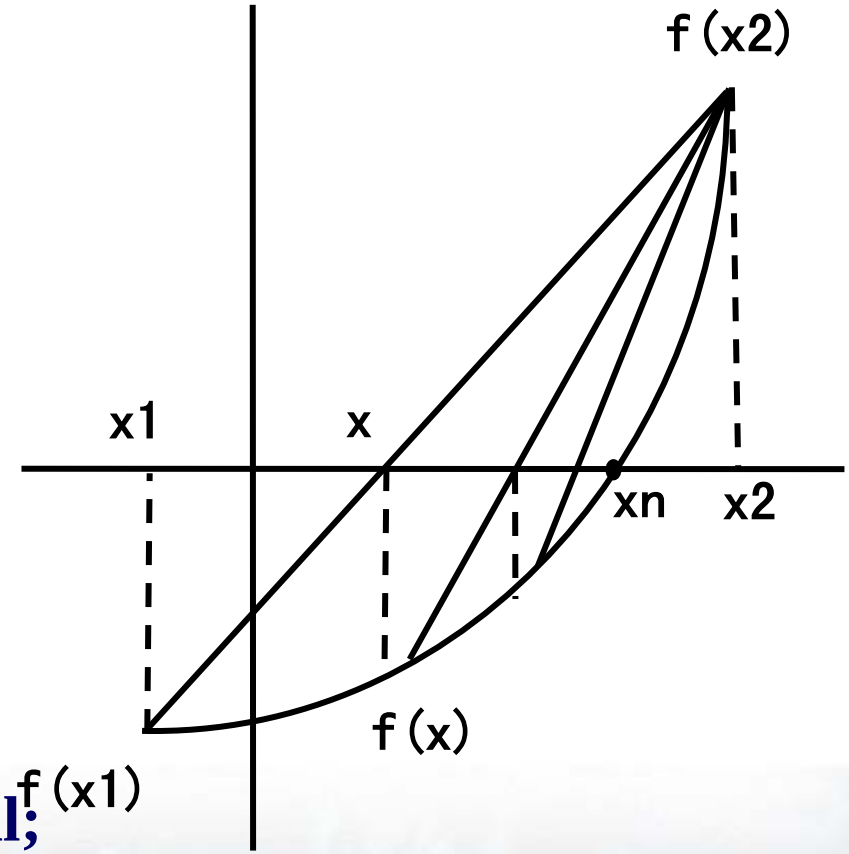
可以组织出的函数

- 求f(x):
  - 输入x，输出f (x);
- xpoint (x1, x2):
  - 输入x1, x2 输出弦与x轴的交点;
- 总方程: root(x1, x2):
  - 输入x1, x2 输出根;



函数嵌套调用程序实例

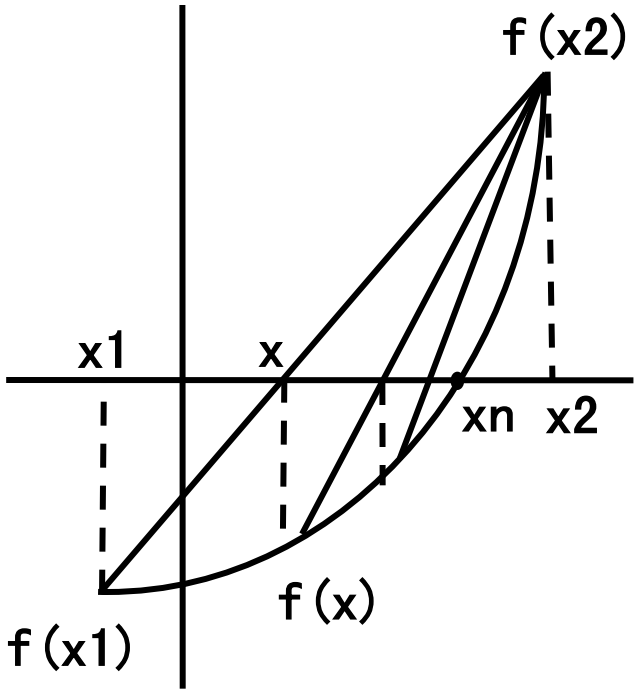
```
#include<iostream>
#include<cmath>
using namespace std;
float Root(float, float);
float XPoint(float, float);
float f(float x);
int main( )
{
    float x1, x2, f1, f2, x;
    do{
          cin>>x1>>x2;
          f1=f(x1); f2=f(x2);
    }while(f1*f2 >= 0);
    x=Root(x1,x2);
    cout<<"The root is"<<x<<endl;
    return 0;
}
```



北京大学

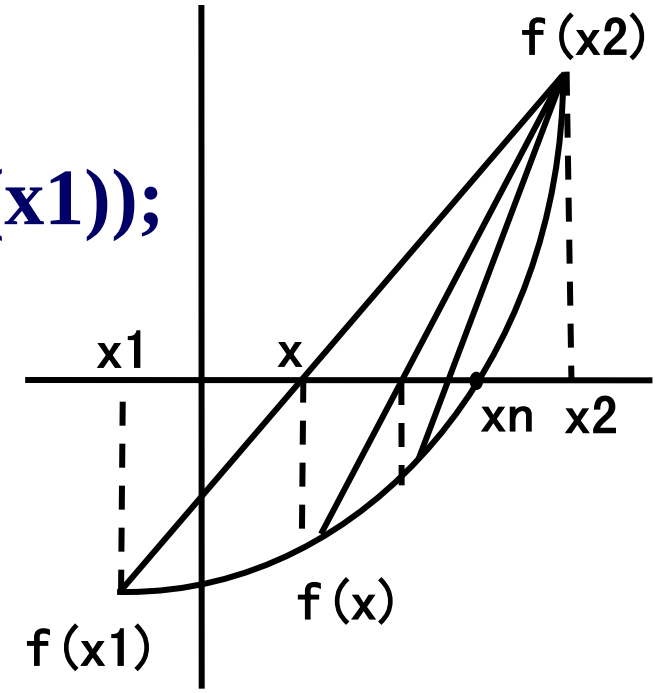
函数嵌套调用程序实例

```
float Root(float x1, float x2)
{
    float x, y, y1;
    y1 = f(x1);
    do {
        x = XPoint(x1, x2);
        y = f(x) ;
        if (y*y1<0)
        {  x1= x; }
        else
                x2 = x;
    } while(fabs(y)>=0.0001);
    return(x);
}
```



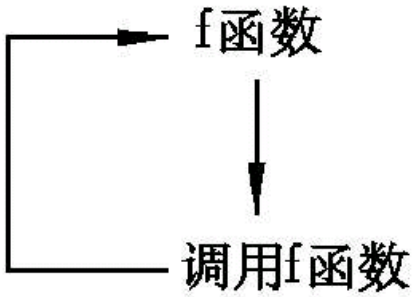
函数嵌套调用程序实例

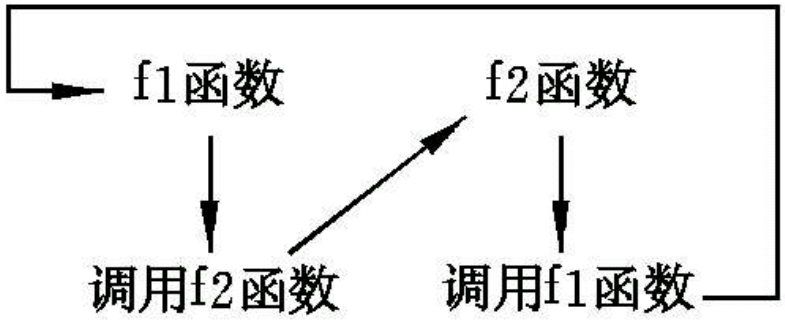
```
float XPoint(float x1, float x2)
{
    float x;
    x=(x1*f(x2)-x2*f(x1))/(f(x2)-f(x1));
    return(x);
}

float f(float x)
{
    float y;
    y=((x-0.5)*x+16.0)*x-80.0;
    return(y);
}
```



函数的递归调用

- C++语言允许递归调用
  - 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。





  - 程序中不应出现无终止的递归调用，必须控制只有在某一条件成立时才继续执行递归调用，否则就不再继续。

例题分析（1）

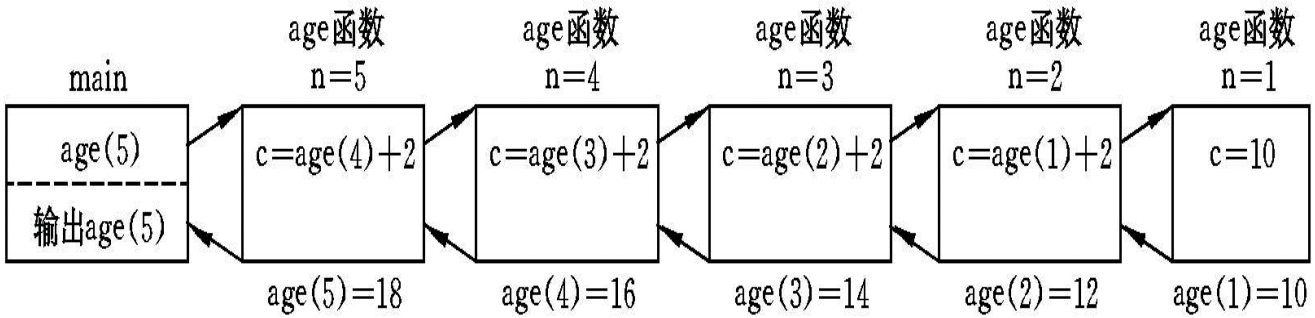
- 有5个人坐在一起，问第5个人多少岁？他说比第4个人大2岁。问第4个人岁数，他说比第3个人大2岁。问第3个人，又说比第2个人大2岁。问第2个人，说比第1个人大2岁。最后问第1个人，他说是10岁。请问第5个人多大。
  - Age[1] = 10;
  - Age[2] = age[1] + 2;
  - Age[3] = age[2] + 2;
  - Age[4] = age[3] + 2;
  - Age[5] = age[4] + 2;

例题分析（1）

```
#include <iostream.h>
int main()
{  int age[6];
   age[1] = 10;
   for (int i = 2; i <= 5; i++)
   {  age[i] = age[i-1] + 2;}
   cout << "第5个人的年龄是:  "<<age[5] <<endl;
   return 0;
}
```

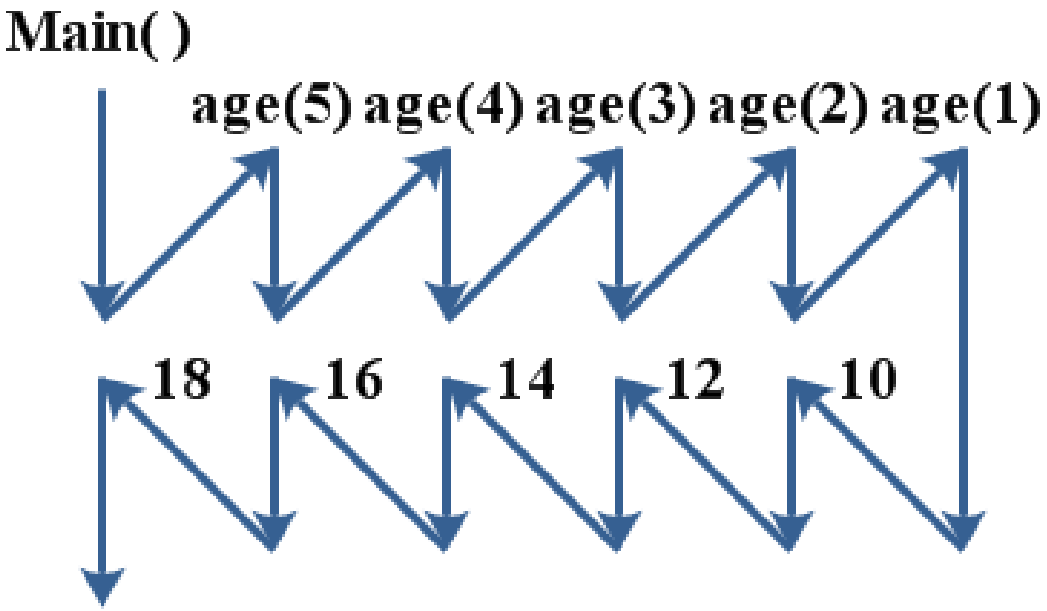
另一种解决方案

```
#include<iostream.h>
int age(int n)
{   int c;
    if(n == 1)
          c = 10;
    else
          c = age(n-1)+2;
    return(c);
}
void main()
{   cout<<“第五个人的年龄是：”<<age(5);
}
```



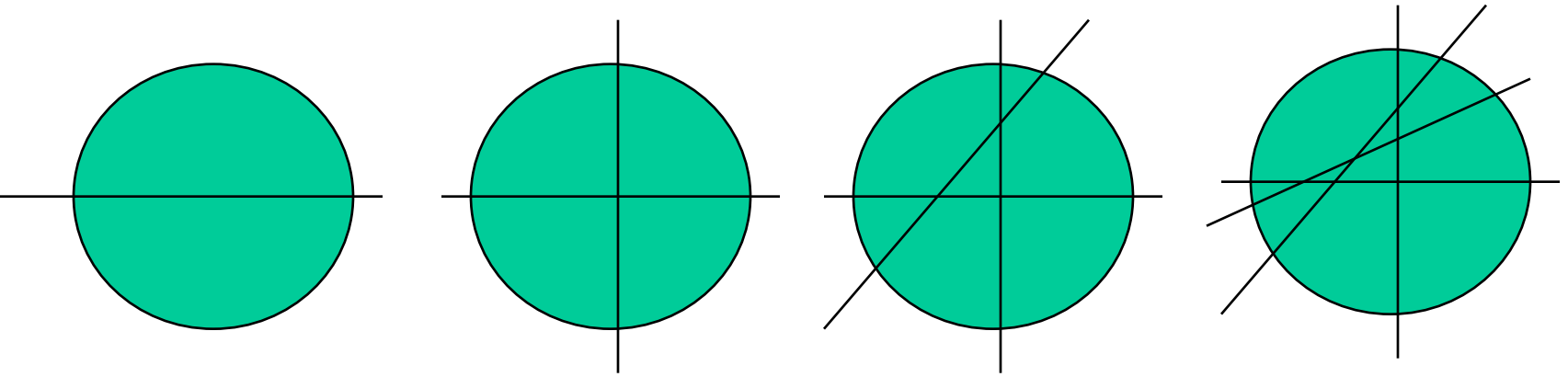
另一种解决方案

```
#include<iostream.h>
int age(int n)
{   int c;
    if(n == 1)
          c = 10;
    else
          c = age(n-1)+2;
    return(c);
}
void main()
{   cout<<“第五个人的年龄是：”<<age(5);
}
```



例题分析（2）

- 切饼，100刀最多能切多少块？



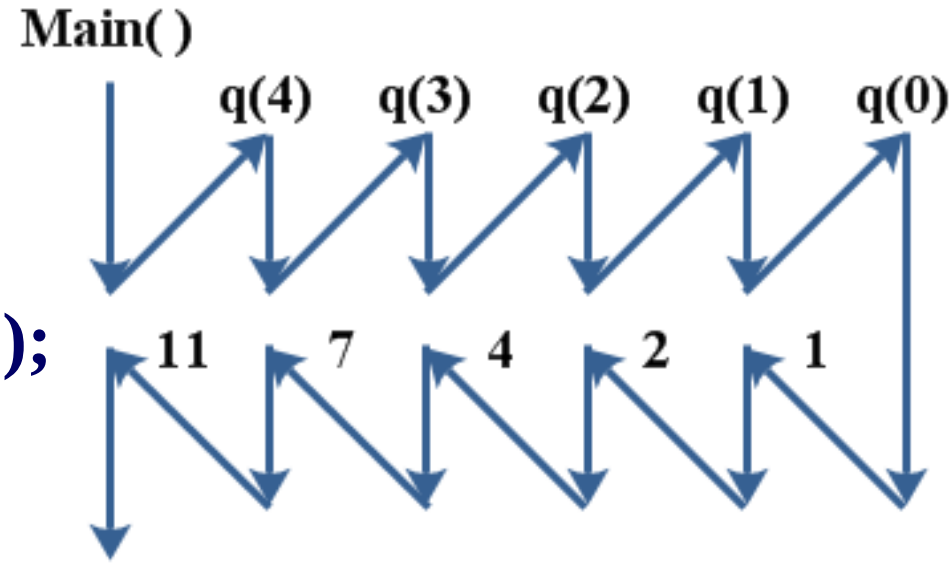
- q(1)=1+1=2
- q(2)=1+1+2=4；
- q(3)=1+1+2+3=7；
- q(4)=1+1+2+3+4=11；
- q(n)=q(n−1)+n；q(0)=1；

例题分析（2）

```
#include <iostream.h>
int main()
{   int q[101];
    q[0] = 1;
    for (int i = 1; i <= 100; i++)
    {  q[i] = q[i-1] + i;}
    cout<<"100刀最多可切"<<q[100]<<"块"<<endl;
    return 0;
}
```

另一种解决方案

```
#include<iostream>
using namespace std;
int q(int n){
    if (n == 0)
        return 1;
    else
        return( n + q(n-1));
}

int main(){
        cout<<q(4)<<endl;
        return 0;
}
```



递推数列

- 一个数列从某一项起，它的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:
  - 1!, 2!, 3!, … n!
    - fact(n) = n * fact(n-1)（通项公式）;
    - fact(1) =1 （边界条件）
  - 1, 1, 2, 3, 5, 8, 13, 21, …
    - fab(n)= fab(n-1) + fab(n-2)（通项公式）;
    - fab(1)=1, fab(2)=1;（边界条件）

递推与递归

- 递推问题
  - 后续的运算依赖于已知的条件；当前的运算是下一步运算的基础；
- 解法1：从已知的初始条件出发，逐次去求所需要的值。

  如求n!

  初始条件 fact(1) = 1

  fact(2) = 2*fact(1) = 2

  fact(3) = 3*fact(2) = 6

  ……

递推与递归

- 递推问题
  - 后续的运算依赖于已知的条件；当前的运算是下一步运算的基础；
- 解法2：递归算法
  - 出发点不放在初始条件上，而放在求解的目标上，从所求的未知项出发逐次调用本身的求解过程，直到递归的边界（即初始条件）。

例题分析（3）

- 问题：已知n，求 n!

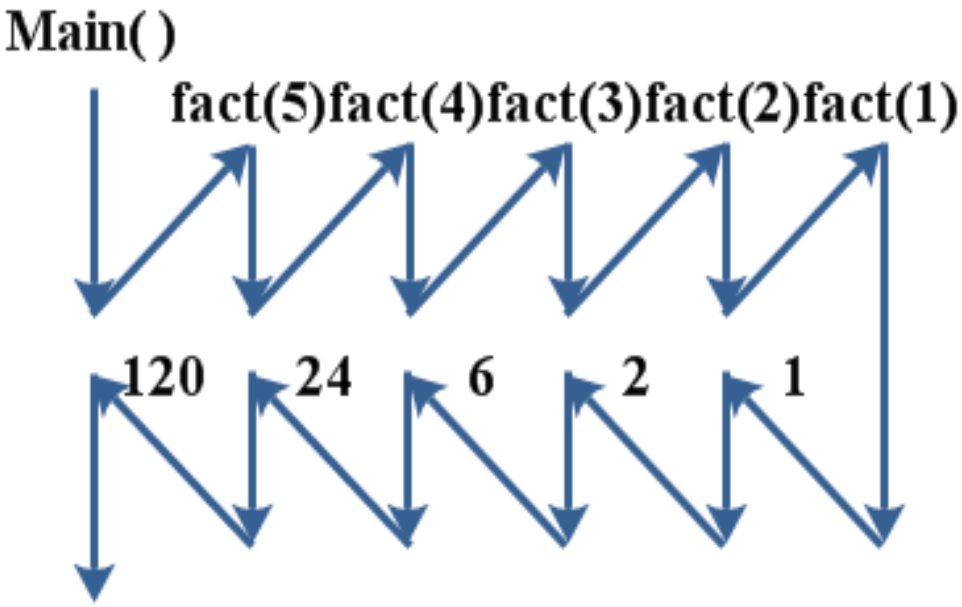
  假设计算阶乘的任务由一个函数fact来做

  fact(n)　　等于　　fact(n-1)*n
  fact(n-1)　等于　　fact(n-2)*(n-1)
  fact(3)　　等于　　fact(2)*3;
  fact(2)　　等于　　fact(1)*2
  fact(1)　　等于　　1;

- 可知
  - fact(n)的值等于fact(n-1)*n;
  - fact(1) = 1;

例题分析（3）

```
#include<iostream>
using namespace std;
int Factorial(int n){
    int temp;
    if(n == 1)
        return 1;
    else{
        temp = Factorial( n-1 );
        temp = n*temp;
        return temp;
    }
}
int main(){
        cout<<Factorial(5)<<endl;
        return 0;
}
```



例题分析（3）


```
(1)  int Factorial(int n);
(2)  {  int temp;
(3)     if(n==0)
(4)         return 1;
(5)     else{
(6)         temp=Factorial(n-1);
(7)         temp=n*temp;
(8)         return temp;
(9)         }
(10)  }
```

本演示只给出函数Factorial(n)执行过程中，系统栈的变化。为方便说明，本演示只对参数n和返回地址进行入栈和出栈操作，局部变量 temp不做入栈和出栈操作。

知道了

递归问题例题

北京大学

程序分析（1）

```
#include<iostream.h>
void function(int num){
    cout<<num%10;
    if ((num/10)!= 0)
        function (num/10);
}
void main(){
    int num;
    cin>>num;
    function (num);
}
```

试分析这个程序执行了什么功能。

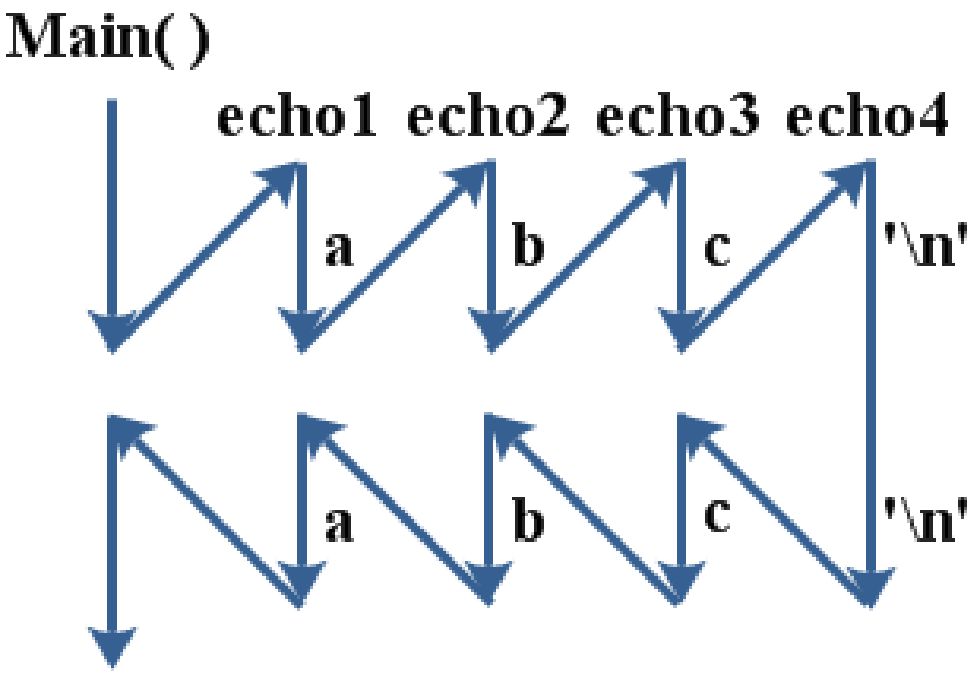
程序分析（2）

```
#include<iostream>
using namespace std;
void echo()
{  char c;
   c = cin.get();
   if (c != '\n')   echo( );
   cout<<c;
}
void main()
{  echo( ); }
```

试分析这个程序执行了什么功能。

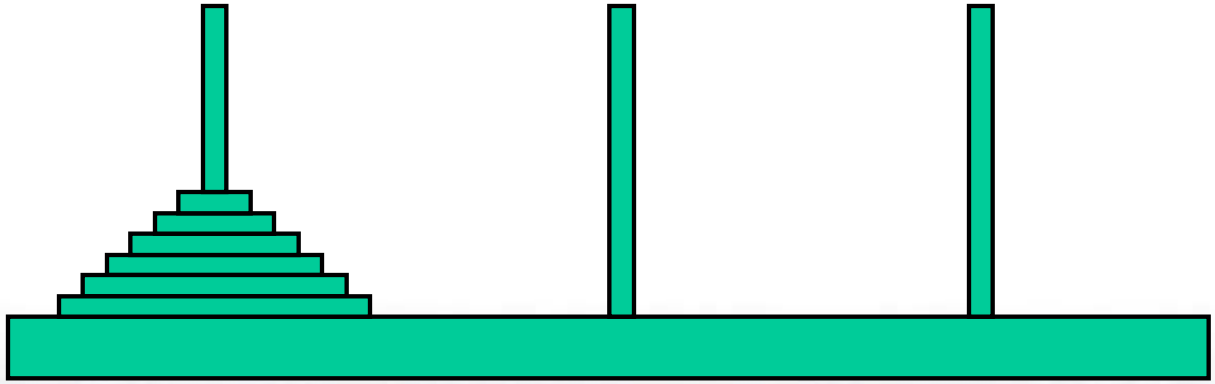
程序分析（2）

```
#include<iostream>
using namespace std;
void echo()
{  char c;
   c = cin.get();
   if (c != '\n')   echo( );
   cout<<c;
}
void main()
{  echo( ); }
```



递归经典问题——汉诺塔问题

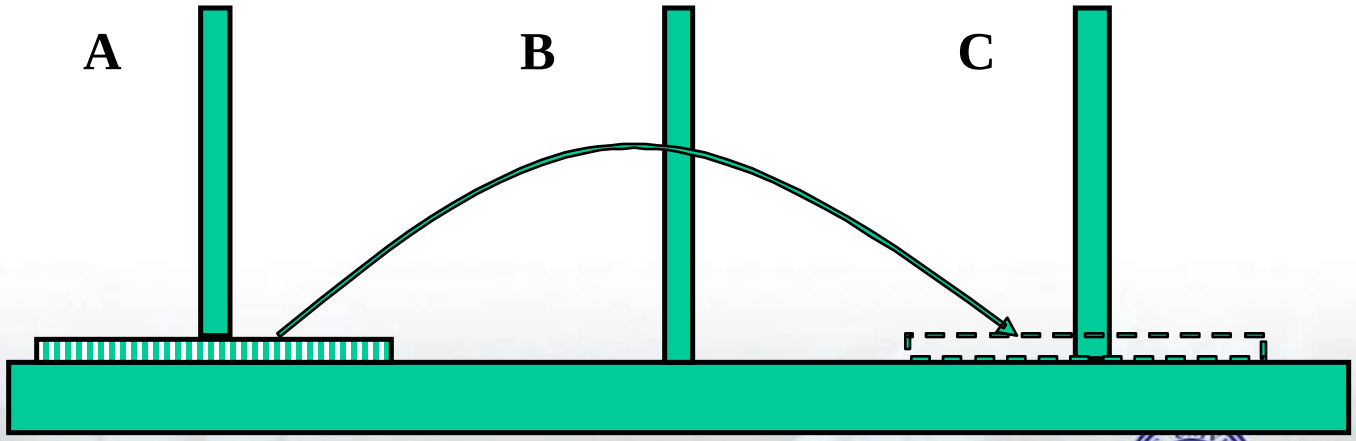
- 故事：相传在古代印度的Bramah庙中，有位僧人整天把三根柱子上的金盘倒来倒去，原来他是想把64个一个比一个小的金盘从一根柱子上移到另一根柱子上去。移动过程中恪守下述规则：每次只允许移动一只盘，且大盘不得落在小盘上面。
- 有人会觉得这很简单，真的动手移盘就会发现，如以每秒移动一只盘子的话，按照上述规则将64只盘子从一个柱子移至另一个柱子上，所需时间约为5800亿年。



递归经典问题——汉诺塔问题

怎样编写这种程序？从思路上还是先从最简单的情况分析起，搬一搬看，慢慢理出思路。

1、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



北京大学

2、在A柱上有二只盘子，1为小盘，2为大盘。

第（1）步将1号盘从A移至B；

第（2）步将2号盘从A移至C；

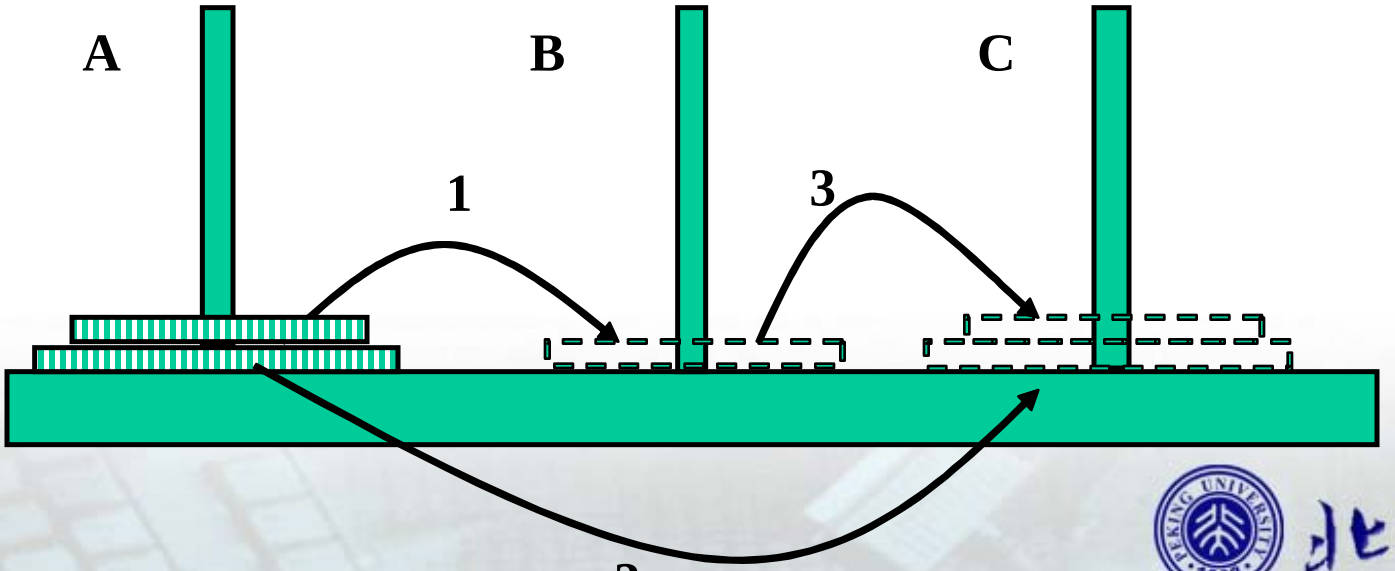
第（3）步再将1号盘从B移至C；

这三步记为：

move 1 from A to B;

move 2 from A to C;

move 3 form B to C;



3、在A柱上有3只盘子，从小到大分别为1号，2号，3号

第（1）步将1号盘和2号盘视为一个整体；先将二者作为整体从A移至B。这一步记为

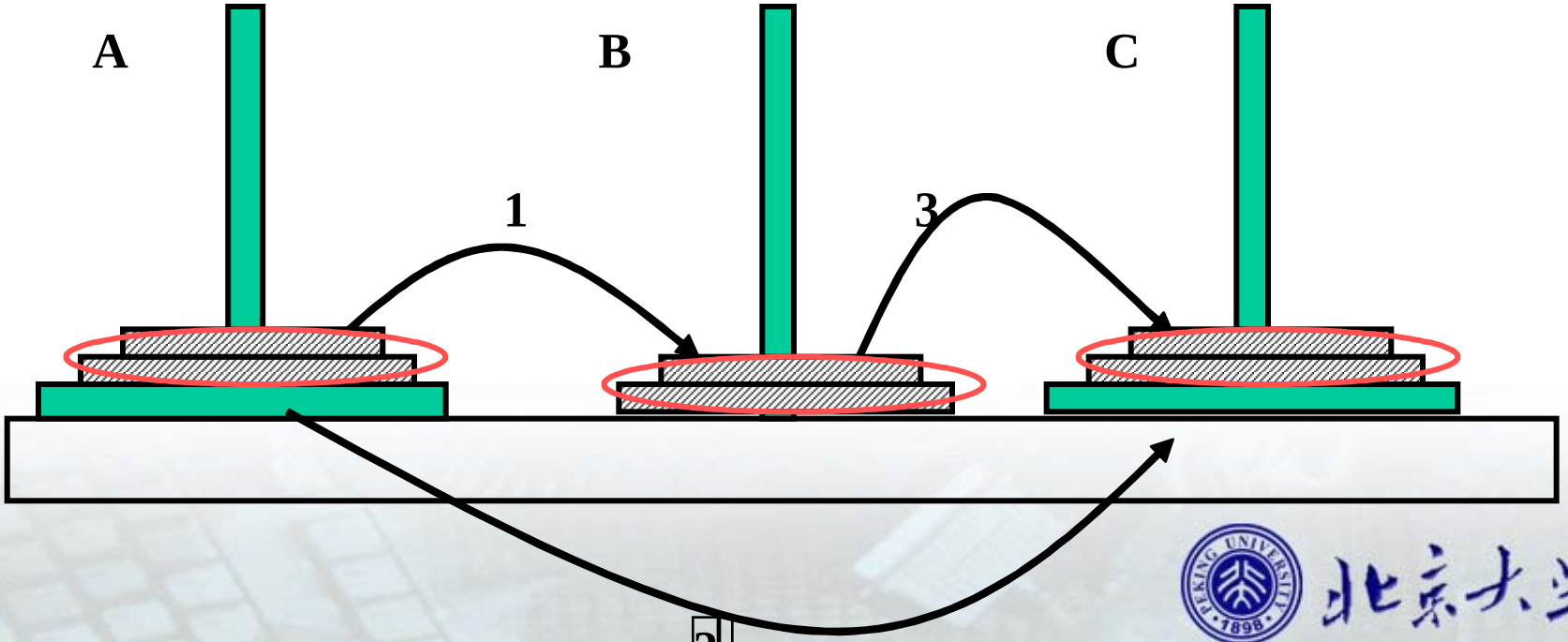
move(2, A, C, B)

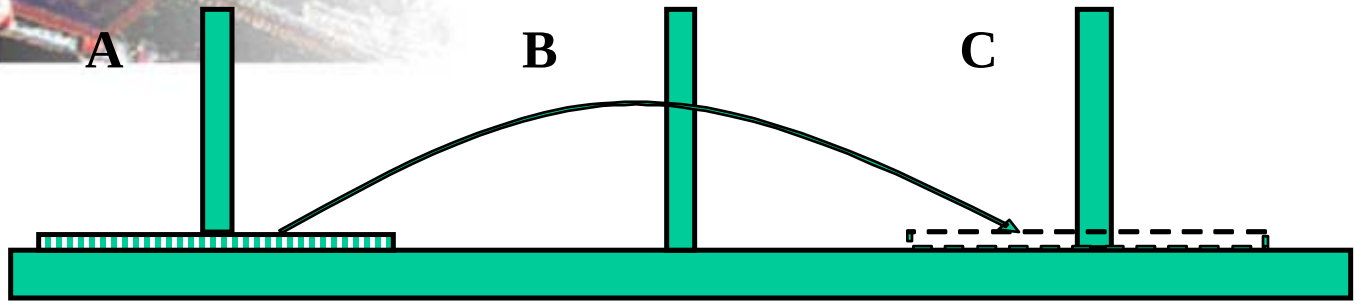
第（2）步将3号盘从A移至C，一次到位。记为

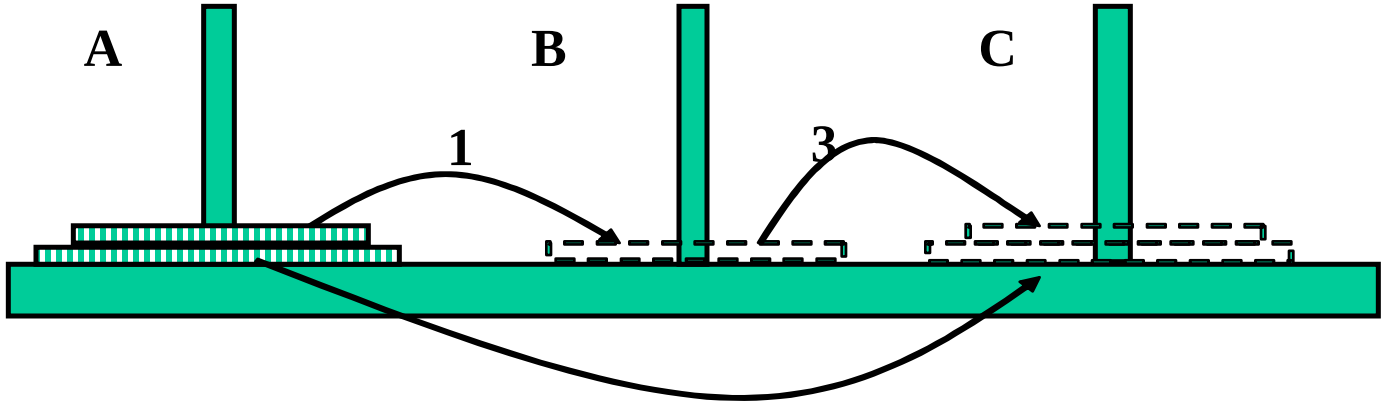
move 3 from A to C

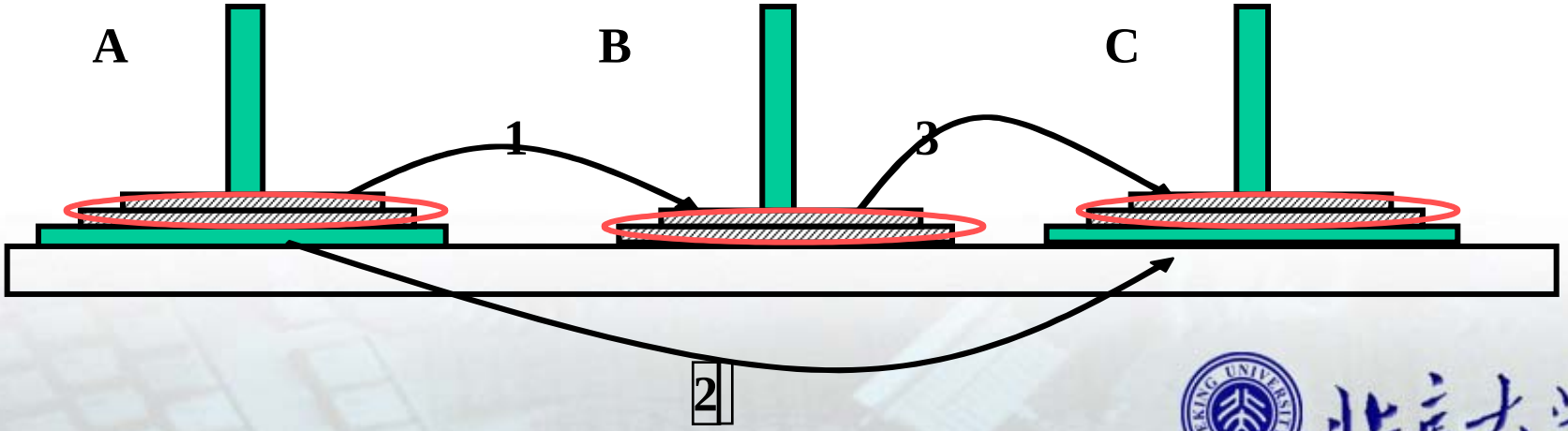
第（3）步处于B上的作为一个整体的2只盘子，再移至C。这一步记为

move(2, B, A, C)




A
B
C


A
B
C
1
3
2


A
B
C
1
3
2

递归经典问题——汉诺塔问题

- move(n, A, B, C) 分解为3步
  - move(n-1, A, C, B)理解为将上面的n-1只盘子作为一个整体从A经C移至B;
  - 输出n: A to C，理解将n号盘从A移至C，是直接可解结点;
  - move(n-1, B, A, C)理解为将上面的n-1只盘子作为一个整体从B经A移至C。

```
#include<iostream>
using namespace std;
void  move(int m, char A, char B, char C) //表示将m个盘子从A经过B移动到C
{   if (m==1)                              //如果m为1,则为直接可解结点
    {
        cout<<"move 1# from"<<A<<" to "<<C<<endl;  //直接可解结点
    }
    else                          //如果不为1, 则要调用move(m-1)
    {
        move(m-1,A,C,B);          //递归调用move(m-1)
        cout<<"move 1# from"<<A<<" to "<<C<<endl;  //直接可解结点
        move(m-1,B,A,C);          //递归调用move(m-1)
    }
}
int main() {
    int n;                                //整型变量, n为盘数
    cout<<"请输入盘数n="<<endl;
    cin >> n;                               //输入盘子数目正整数n
    cout<<"在3根柱子上移"<<n<<"只盘的步骤为:"<<endl;
    move(n,'a','b','c');                  //调用函数 move(n,'a','b','c')
    return 0;
}
```

数 独

7	1	0	0	0	5	0	9	6
0	0	9	2	6	0	0	3	4
0	0	2	9	0	0	8	0	0
5	0	0	0	4	0	6	2	3
1	0	0	7	0	3	0	0	9
0	8	3	0	5	0	0	0	7
0	0	5	0	0	8	3	0	0
4	3	0	0	7	2	9	0	0
9	0	0	4	0	0	0	5	1

北京大学

好好想想,有没有问题?

谢　谢　！

北京大学