西北工业大学
NORTHWESTERN POLYTECHNICAL UNIVERSITY

C程序设计

Programming in C

1011014

主讲：姜学锋，计算机学院

调用函数 - 调用形式

1、内联函数

2、函数嵌套和递归调用

4.4 内联函数

- 对于一些函数体代码不是很大，但又频繁地被调用的函数，准备执行函数的时间竟然比函数执行的时间要多很多。

4.4 内联函数

- 新版本的C语言标准提供一种提高函数效率的方法，即在编译时将被调函数的代码直接嵌入到主调函数中，取消调用这个环节。这种嵌入到主调函数中的函数称为内联函数（inline function）。

- 一些早期编译器，如Visual C++ 6.0不支持这个新特性。

4.4 内联函数

- 内联函数的声明是在函数定义的类型前加上inline修饰符，定义形式为：

```
inline 返回类型 函数名(形式参数列表)
{
   函数体
}
```

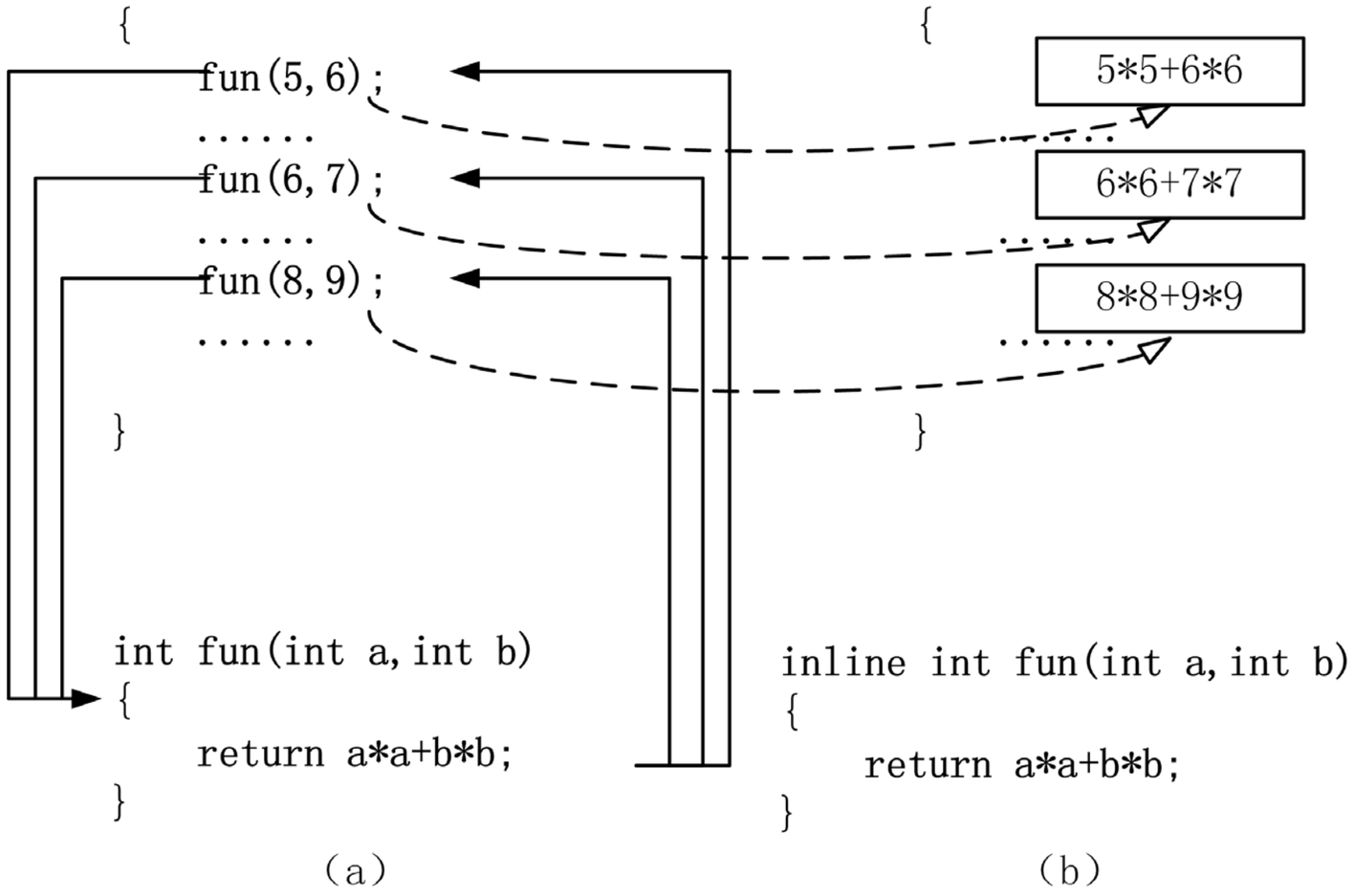
- 或在函数原型的返回类型前加上inline修饰符，声明形式为：

```
inline 返回类型 函数名(类型1 参数名1,类型2 参数名2,...);
```

4.4 内联函数

- 内联函数可以同时在函数定义和函数原型中加inline修饰符，也可以只在其中一处加inline修饰符，但内联的声明必须出现在内联函数第一次被调用之前。

图4.2 普通函数和内联函数调用示意



- 所以内联函数的优点是从源代码层面看，有函数的结构；而在编译后，却没有函数的调用开销（已不是函数了）。

4.4 内联函数

【例4.6】

计算两个数的平方和。

4.4 内联函数

例4.6

```
1 #include <stdio.h>
2 inline int fun(int a,int b) //内联函数
3 {
4   return a*a+b*b;
5 }
6 int main()
7 {
8   int n=5,m=8,k;
9   k = fun(n,m); //调用点嵌入 a*a+b*b 代码
10  printf("k=%d\n",k);
11  return 0;
12 }
```

4.4 内联函数

- 使用内联函数就没有函数的调用了，因而就不会产生函数来回调用的效率问题。

- 但是由于在编译时函数体中的代码被嵌入到主调函数中，因此会增加目标代码量，进而增加空间开销。可见内联函数是以目标代码的增加为代价来换取运行时间的节省。

4.4 内联函数

- 内联函数中不允许用循环语句和switch语句，递归函数也不能被用来做内联函数。当编译器无法对代码进行嵌入时，就会忽略inline声明，此时内联失效，这些函数将按普通函数处理。

- 一般情况下，只是将规模较小、语句不多（1～5个）、频繁使用的函数声明为内联函数。对一个含有许多语句的函数，函数调用的开销相对来说微不足道，所以也没有必要用内联函数实现。

4.5 函数调用形式

- C语言函数调用形式有两种：
- (1)嵌套调用；
- (2)递归调用；

4.5.1 嵌套调用

- 在调用一个函数的过程中，又调用另一个函数，称为函数嵌套调用，C语言允许函数多层嵌套调用，只要在函数调用前有函数声明即可。

4.5.1 嵌套调用

【例4.7】

函数嵌套调用示例。

4.5.1 嵌套调用

例4.7

```
1 #include <stdio.h>
2 int fa(int a,int b); //fa函数原型
3 int fb(int x); //fb函数原型
4 int main()
5 {
6   int a=5,b=10,c;
7   c = fa(a,b);
8   printf("%d\n",c);
9   c = fb(a+b);
10  printf("%d\n",c);
11  return 0;
12 }
13 int fa(int a,int b)
14 {
15  int z;
```

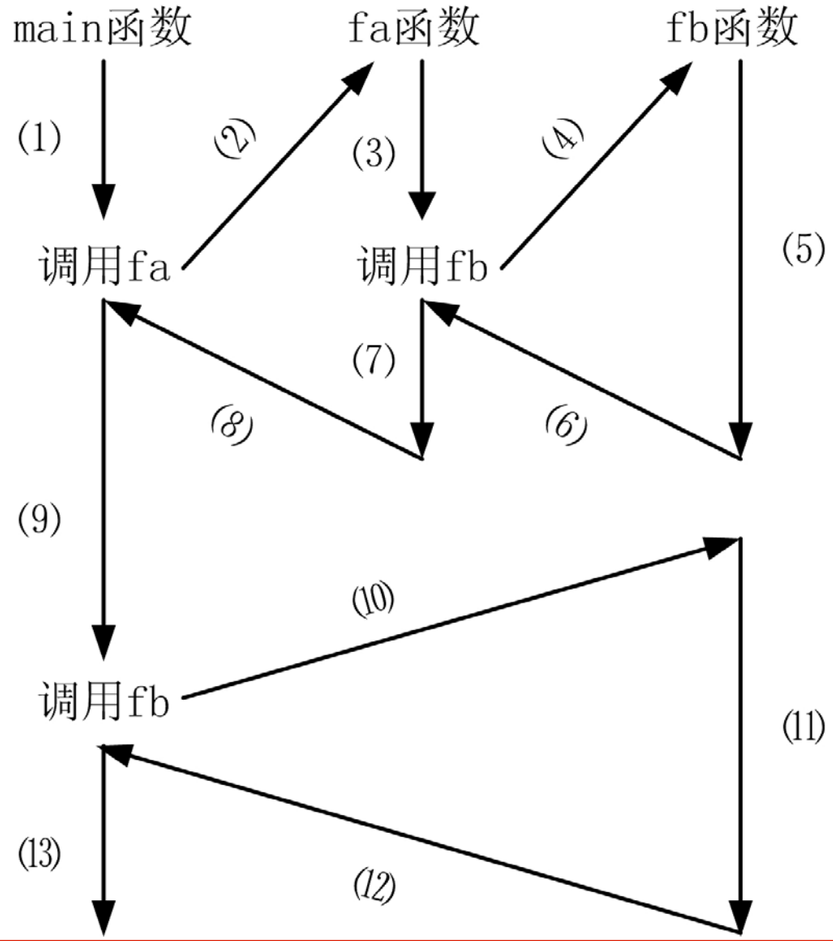
4.5.1 嵌套调用

例4.7

```
16    z= fb(a*b);
17    return z;
18 }
19 int fb(int x)
20 {
21    int a=15,b=20,c;
22    c=a+b+x;
23    return c;
24 }
```

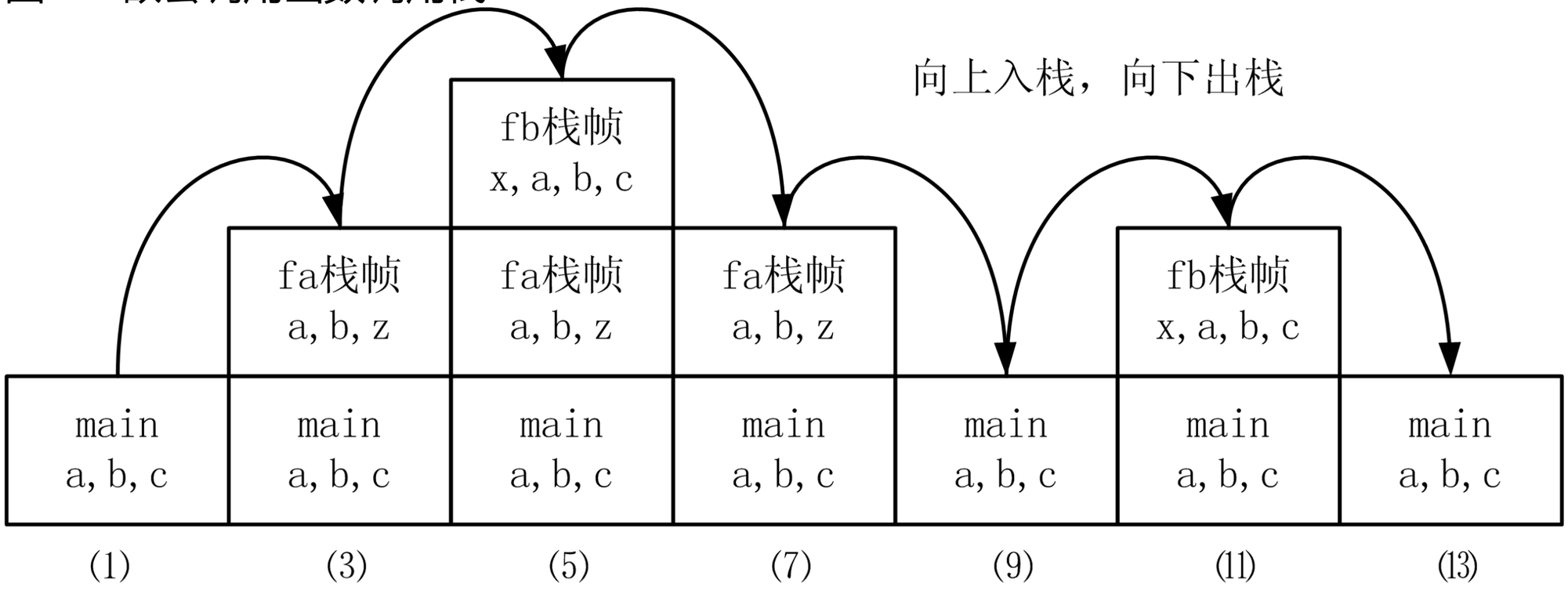
4.5.1 嵌套调用

图4.3 嵌套调用示意



4.5.1 嵌套调用

图4.4 嵌套调用函数调用栈



方框内第一行说明是哪个函数栈帧，第二行说明该栈帧有哪些形参和变量

4.5.1 嵌套调用

- 从图中可以看到：
- （1）函数每调用一次就会有新的函数调用栈建立，返回时函数调用栈释放；
- （2）每个函数调用栈都是独立的，相互不影响；
- （3）尽管main函数和fb函数都有局部变量a、b、c，但明显的是它们是在不同区域的存储单元，各自独立，互不相干。

4.5.1 嵌套调用

【例4.8】

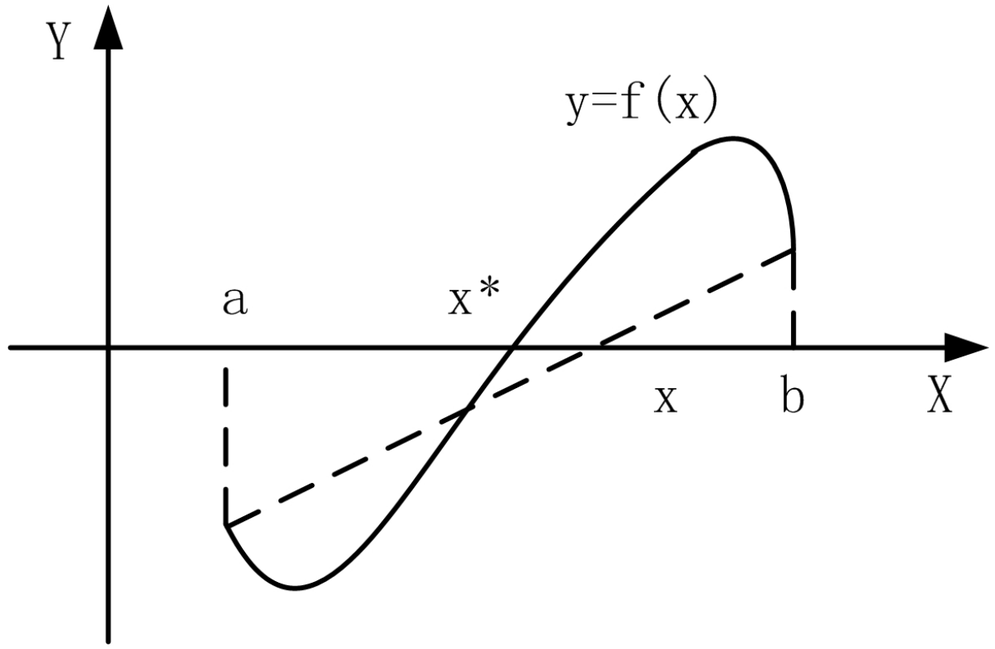
用弦截法求方程 $f(x)=x^3-5x^2+16x-80$ 的根，精度 $\varepsilon=10^{-6}$ 。

4.5.1 嵌套调用

例题分析

如图4.5所示，设$f(x)$ 在$[a,b]$ 内连续， 在$f(x)=0$ 内有单根x^* 。

图4.5 f(x)函数曲线



4.5.1 嵌套调用

例题分析

用双点弦截法求 $f(x)$ 在$[a,b]$的单根x^*的方法是：$f(x)=0$

①过点$(a,f(a))$,$(b,f(b))$ 作一条直线，与X轴相交，设交点横坐标为 $\tilde{x}$；

②若 $f(\tilde{x})=0$ ，则$\tilde{x}$ 为精确根，迭代结束；否则判断根x^*在$\tilde{x}$ 的哪一侧，排除 $[a,b]$中没有根 x^* 的一侧，以 $\tilde{x}$为新的有根区间边界，得到新的有根区间，仍记为 $[a,b]$ 。循环②步。

③计算 $\tilde{x}$ 的公式为

$$\tilde{x}=b-f(b)\frac{b-a}{f(b)-f(a)}=\frac{af(b)-bf(a)}{f(b)-f(a)}$$

4.5.1 嵌套调用

例题分析

（1）设计函数f计算 $f(x)$；

（2）设计函数root求 $[a,b]$ 的根。

4.5.1 嵌套调用

例4.8

```
 1 #include <stdio.h>
 2 #include <math.h>
 3 double f(double x)
 4 { //所要求解的函数公式，可改为其他公式
 5   return x*x*x-3*x-1;
 6 }
 7 double point(double a,double b)
 8 { //求解弦与x轴的交点
 9   return (a*f(b)-b*f(a))/(f(b)-f(a));
10 }
11 double root(double a, double b)
12 { //弦截法求方程[a,b]区间的根
13   double x,y,y1;
14   y1=f(a);
15   do {
```

4.5.1 嵌套调用

例4.8

```
16      x=point(a,b); //求交点x坐标
17      y=f(x); //求y
18      if (y*y1>0) y1=y, a=x;
19      else b=x;
20    } while (fabs(y)>=0.00001); //计算精度E
21    return x;
22 }
23 int main()
24 {
25    double a,b;
26    scanf("%lf%lf",&a,&b);
27    printf("root=%lf\n",root(a,b));
28    return 0;
29 }
```

4.5.1 嵌套调用

例4.8

```
1 #include <stdio.h>
```

程序运行屏幕

```
root=1.879385
```

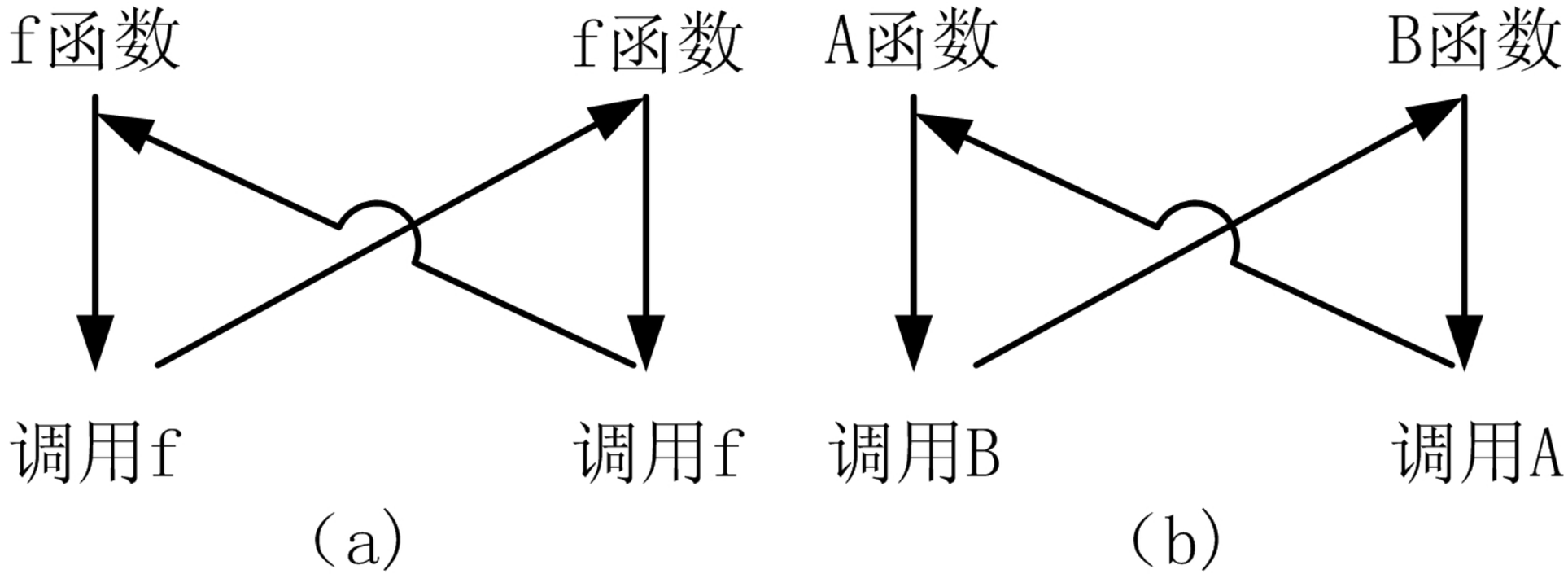
```
1 2↙
```

4.5.2 递归调用

- 函数直接或间接调用自己称为递归调用。C语言允许函数递归调用，如图（a）所示为直接递归调用，如图（b）所示为间接递归调用。

4.5.2 递归调用

图4.6 递归调用示意



4.5.2 递归调用

【例4.9】

编写求n的阶乘的函数。

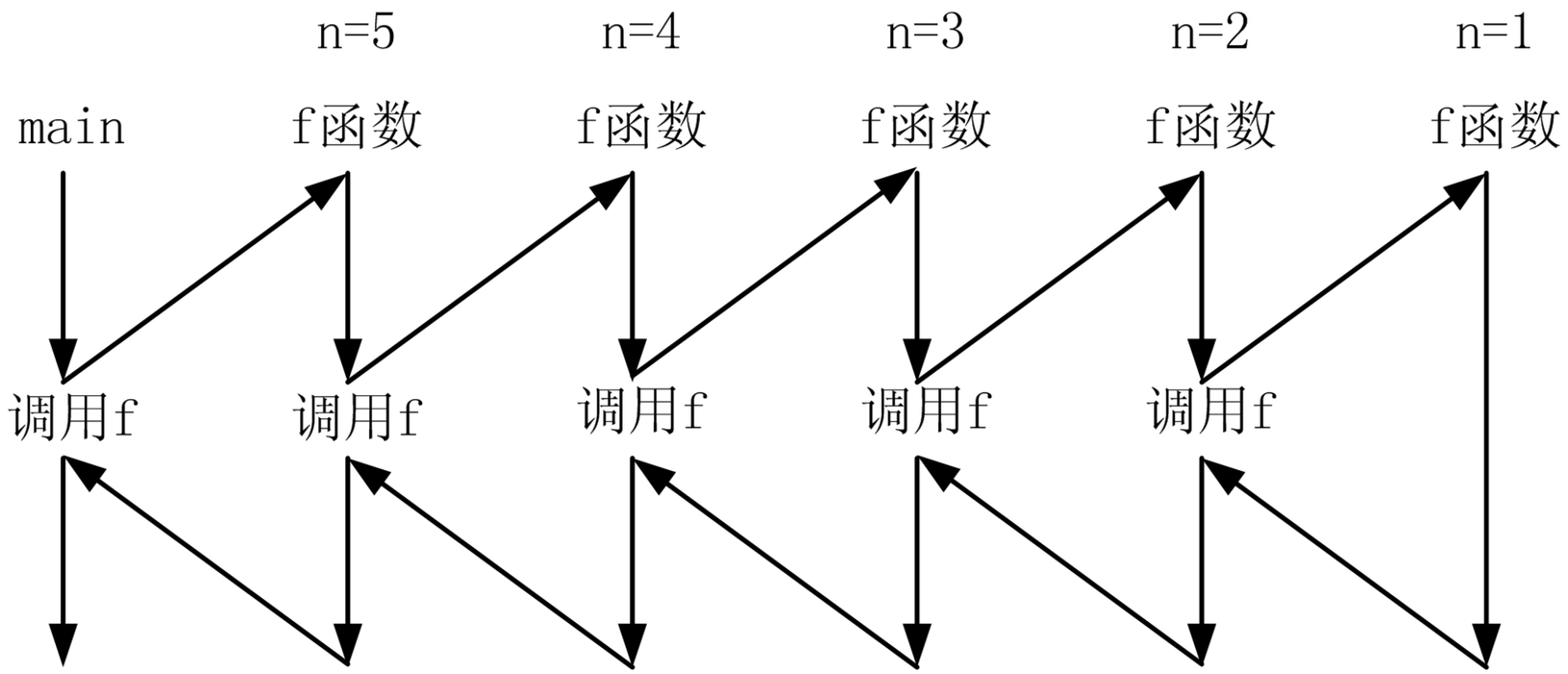
4.5.2 递归调用

例4.9

```
1 #include <stdio.h>
2 int f(int n)
3 {
4   if (n>1) return f(n-1)*n; //递归调用
5   return 1;
6 }
7 int main()
8 {
9   printf("%d\n",f(5));
10  return 0;
11 }
```

4.5.2 递归调用

图4.7 递归调用过程



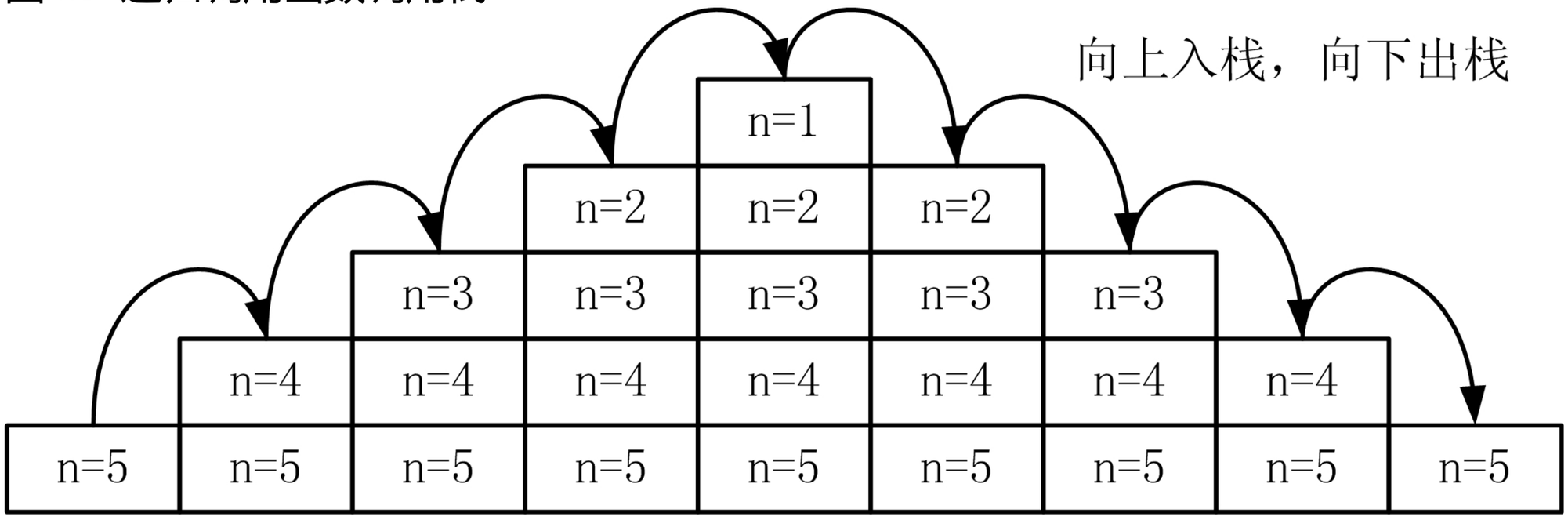
4.5.2 递归调用

表4-2 f函数的执行跟踪

n	return
5	f(4)*5
4	f(3)*4
3	f(2)*3
2	f(1)*2
1	1

4.5.2 递归调用

图4.8 递归调用函数调用栈



仅示意f栈帧的情况，方框内是形参n。

4.5.2 递归调用

- 从图中可以看到，在函数递归调用时，递归函数每次调用其本身，一个新的函数栈就会被使用，这个新函数栈里的形参、变量和该函数的另一个函数栈里面的形参、变量是完全不同的内存单元。

4.5.2 递归调用

- 从图中还可以看到，递归函数必须定义一个终止条件，否则函数会永远递归下去，直到栈空间耗尽。所以，递归函数内一般都用类似if语句来判定终止条件，如果条件成立则继续递归调用，否则函数结束递归开始返回。

4.5.2 递归调用

```c
//①程序A
#include <stdio.h>
void f(int n) //递归处理顺序
{
  printf("%d->",n);
  if (n>1) f(n-1) ;
}
int main()
{
  f(5);
  return 0;
}
```

```c
//②程序B
#include <stdio.h>
void f(int n) //递归处理顺序
{
  if (n>1) f(n-1) ;
  printf("%d->",n);
}
int main()
{
  f(5);
  return 0;
}
```

程序A和程序B仅仅是“printf(”%d->“,n)”执行顺序的不同，程序A运行的结果是“5->4->3->2->1->”，程序B运行的结果是“1->2->3->4->5->”。


程序设计