

§ 4. 利用函数实现指定的功能

4.2. 函数的嵌套调用

4.2.1. C++程序的执行过程 (P. 93 9步)

- (1) 执行main函数的开头部分
- (2) 遇到调用a函数的语句，流程转去a函数
- (3) 执行a函数的开头部分
- (4) 遇到调用b函数的语句，流程转去b函数
- (5) 执行b函数，如果再无其他嵌套的调用，则完成b函数的全部操作
- (6) 返回原来调用b函数的位置，即返回a函数
- (7) 继续执行a函数中尚未执行的部分，直到a函数结束
- (8) 返回main中调用a函数的位置
- (9) 继续执行main函数的剩余部分直到结束

例：程序如下

```
void b()  
{  
    ...  
}  
void a()  
{  
    ...  
    b();  
    ...  
}  
int main()  
{  
    ...  
    a();  
    ...  
    return 0;  
}
```

如何返回？

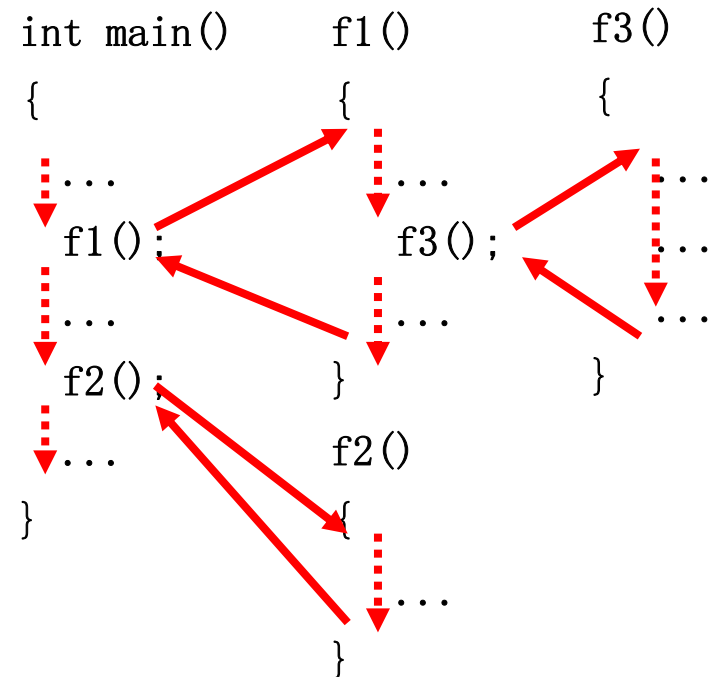


§ 4. 利用函数实现指定的功能

4.2. 函数的嵌套调用

4.2.1. C++程序的执行过程 (通用描述)

- (1) 从main函数的第一个执行语句开始依次执行
- (2) 若执行到函数调用语句，则保存调用函数当前的一些系统信息 (保存现场)
- (3) 转到被调用函数的第一个执行语句开始依次执行
- (4) 被调用函数执行完成后返回到调用函数的调用处，恢复调用前保存的系统信息 (恢复现场)
- (5) 若被调用函数中仍有调用其它函数的语句，则嵌套执行步骤 (2) - (4)
- (6) 所有被调用函数执行完后，顺序执行main函数的后续部分直到结束



4.2.2. 特点

- ★ 嵌套的层次、位置不限
- ★ 遵循后进先出的原则 (栈)
- ★ 调用函数时，被调用函数与其所调用的函数的关系是透明的，适用于大程序的分工组织

§ 4. 利用函数实现指定的功能

4.2. 函数的嵌套调用

4.2.3. 实例

例1：给出求两个整数最大值的函数max2，求4个整数的最大值（多种方法）

```
int main()
{
    int a,b,c,d,m;

    ...输入a/b/c/d四个数字
    m = max4(a,b,c,d);
    ...输出最大值

    return 0;
}
```

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

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

```
int main()
{
    int a,b,c,d,m;

    ...输入a/b/c/d四个数字
    m = max4(a,b,c,d);
    ...输出最大值

    return 0;
}
```

```
int max4(int a, int b,
          int c, int d)
{
    int m1, m2, m;
    m1 = max2(a, b);
    m2 = max2(c, d);
    m = max2(m1, m2);
    return m;
} //改进
```

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

```
int main()
{
    ...
    m = max2( max2( max2(a,b), c), d );
    ...
}
```

一个函数的返回值做为
另一个函数的参数
(本例中函数名相同)

```
int main()
{
    ...
    m = max2( max2(a,b), max2(c,d) );
    ...
}
```

§ 4. 利用函数实现指定的功能

4.2. 函数的嵌套调用

4.2.3. 实例

例1：给出求两个整数最大值的函数max2，求4个整数的最大值（多种方法）

思考：三个函数，6种排列方式，每种方式下最低限度的函数声明？）

```
int max2(int a, int b);  
int max4(int a, int b, int c, int d);
```

```
int main() { }  
int max2(...) { }  
int max4(...) { }
```

```
int max2(...) { }  
int max4(...) { }  
int main() { }
```

```
int max4(...) { }  
int main() { }  
int max2(...) { }
```

```
int main() { }  
int max4(...) { }  
int max2(...) { }
```

```
int max2(...) { }  
int main() { }  
int max4(...) { }
```

```
int max4(...) { }  
int max2(...) { }  
int main() { }
```

§ 4. 利用函数实现指定的功能

4.2. 函数的嵌套调用

4.2.3. 实例

例2: 写一个函数, 判断某正整数是否素数

```
#include <iostream>
#include <cmath>
using namespace std;
int prime(int n)
{
    int i;
    int k = sqrt(n);
    for(i=2; i<=k; i++)
        if (n%i == 0) //两个=
            break;
    return i<=k ? 0 : 1;
}
int main()
{
    int n;
    cin >> n; //为简化讨论, 此处假设输入正确
    cout << n << (prime(n) ? "是":"不是") << "素数" << endl;
    return 0;
}
```

循环的结束有两个可能性:
1、表达式2 ($i \leq k$) 不成立
2、因为 break 而结束

for的表达式2, 直接写成
 $i \leq \text{int}(\text{sqrt}(n))$
是否可以?

§ 4. 利用函数实现指定的功能

4.2. 函数的嵌套调用

4.2.3. 实例

例2：写一个函数，判断某正整数是否素数

03模块例题：找出100-200间的全部素数

```
for(m=101; m<=200; m+=2) { //偶数没必要判断
    prime=true;           //对每个数，先认为是素数
    k=int(sqrt(m)); // k=sqrt(m) 也可(有警告)
    for(i=2; i<=k; i++)
        if (m%i==0) {
            prime=false;
            break;
        }
    if (prime) {
        cout << setw(5) << m;
        n=n+1;           //计数器，只为了加输出换行
        if (n%10==0)      //每10个数输出一行，已改
            cout<<endl;
    }
} //本程序中prime是bool型变量
```

改写为用prime函数

```
int prime(int n)
{
    int i;
    int k = sqrt(n);
    for(i=2; i<=k; i++)
        if (n%i == 0)
            break;
    return i<=k ? 0 : 1;
}

int main()
{
    int m, ret;
    for(m=101; m<=200; m+=2) {
        if (prime(m)) {
            ...//打印
        }
    }
    return 0;
}

//本程序中，prime是函数名
```

§ 4. 利用函数实现指定的功能

4.2. 函数的嵌套调用

4.2.3. 实例

例3: 验证哥德巴赫猜想

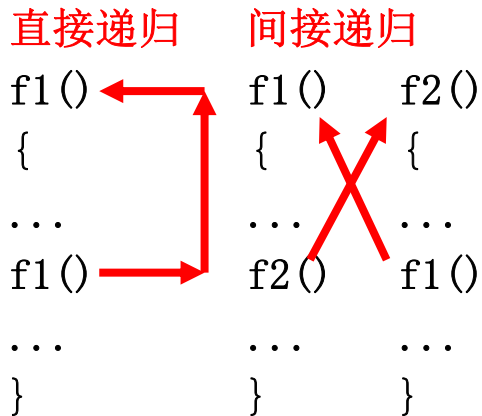
```
#include <iostream>
#include <cmath>
using namespace std;
int prime(int n)
{
    int i;
    int k = sqrt(n);
    for(i=2; i<=k; i++)
        if (n%i == 0) //两个=      一道题目的解可用于另一题中
            break;          强调过程的积累、经验的积累
    return i<=k ? 0 : 1;
}
void gotbaha(int even)
{
    int x;
    for (x=3; x<=even/2; x+=2)
        if (prime(x)+prime(even-x)==2) {
            cout<<x<<"+"<<even-x<<"="<<even<<endl;
            break; //不要break则求出全部组合
        }
}
int main()
{
    int n;
    cin >> n; //为简化讨论, 此处假设输入正确
    gotbaha(n);
    return 0;
}
```

§ 4. 利用函数实现指定的功能

4.3. 函数的递归调用

4.3.1. 含义

函数直接或间接地调用本身



必然有条件判断是否进行下次递归调用!!!

★ 函数的返回值做本函数的参数，是嵌套，不是递归

```
int main()
{
    ...
    m = max2( max2(a, b), max2(c, d));
    ...
}
```

4.3.2. 递归的求解过程

回推：到一个确定值为止 (递归不再调用)

递推：根据回推得到的确定值求出要求的解

例：求解第5个学生的年龄

题目描述：共5个学生

问第5个学生几岁，答：我比第4个大2岁；
问第4个学生几岁，答：我比第3个大2岁；
问第3个学生几岁，答：我比第2个大2岁；
问第2个学生几岁，答：我比第1个大2岁；
问第1个学生几岁，答：我10岁；

回溯 递推

```
age(5) = age(4) + 2;
age(4) = age(3) + 2;
age(3) = age(2) + 2;
age(2) = age(1) + 2;
age(1) = 10;
```


§ 4. 利用函数实现指定的功能

4.3. 函数的递归调用

4.3.3. 如何写递归函数

★ 确定递归何时终止

★ 假设第 $n-1$ 次调用已求得确定值，确定第 n 次调用和第 $n-1$ 次调用之间存在的逻辑关系

=> 不要全面考虑 $1..n$ 之间的变换关系，而应理解为只有 n 和 $n-1$ 两层，且第 $n-1$ 层数据已求得

例1：求解5个学生的年龄

```
int age(int n)
{
    if (n==1)
        return 10;
    else
        return age(n-1)+2;
}
```

```
int main()
{
    cout << age(5) << endl;
    return 0;
}
```

$\text{age}(5) = \text{age}(4) + 2;$

$\text{age}(4) = \text{age}(3) + 2;$

$\text{age}(3) = \text{age}(2) + 2;$

$\text{age}(2) = \text{age}(1) + 2;$

$\text{age}(1) = 10;$

§ 4. 利用函数实现指定的功能

4.3. 函数的递归调用

4.3.3. 如何写递归函数

例2：采用非递归法和递归法两种方式求解n!

非递归法：

全面考虑1-n的关系，

可得出下列公式：

$$n! = 1*2*\dots*n;$$

```
int fac(int n)
{
    int s=1, i;
    for(i=1; i<=n; i++)
        s = s*i;
    return s;
}
```

递归法：

不全面考虑1-n的关系，

仅考虑n和n-1两层，

且假设n-1层已知

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

$$(n-1)! = (n-1) * (n-2)!$$

...

$$1! = 1$$

$$0! = 1;$$

```
int main() //也可以由键盘输入n值，此处略
{
    int n = 5;
    cout << n << "!=" << fac(5) << endl;
}
```

```
int fac(int n)
{
    if (n==0 || n==1)
        return 1;
    else
        return fac(n-1)*n;
}
```

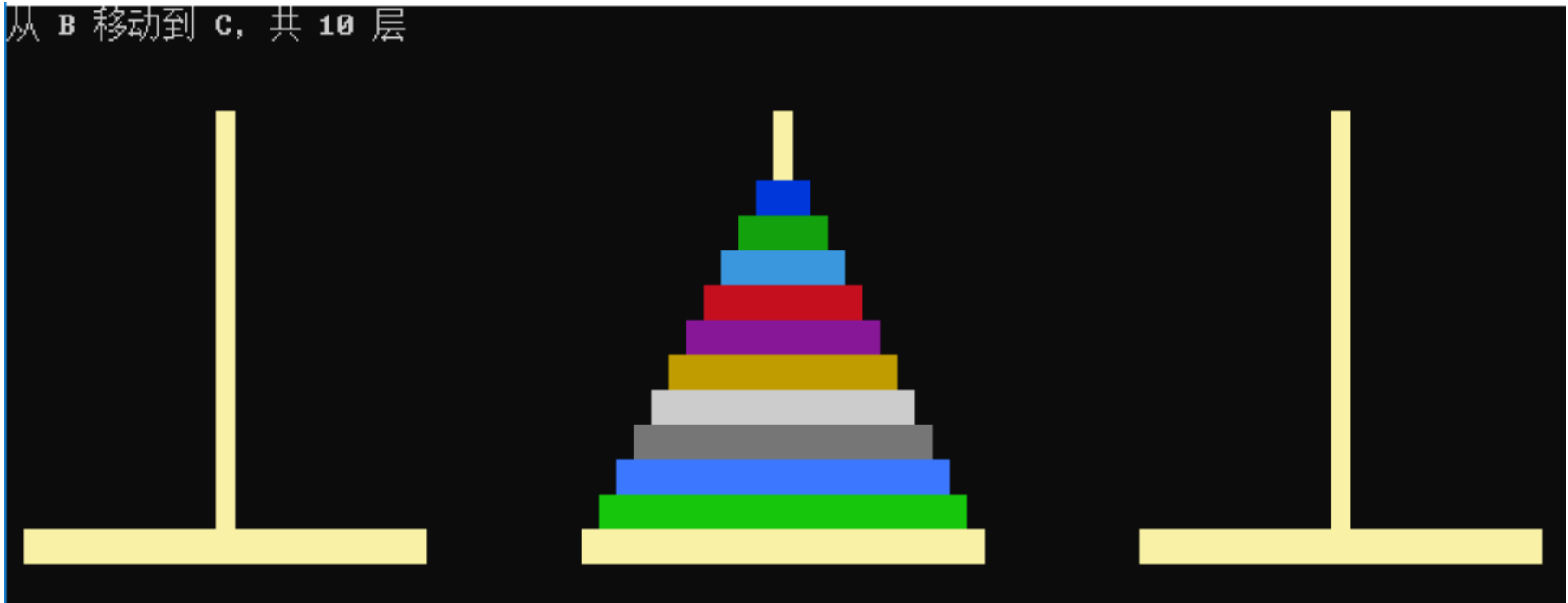
§ 4. 利用函数实现指定的功能

4. 3. 函数的递归调用

4. 3. 3. 如何写递归函数

例3：汉诺塔问题

从 B 移动到 C，共 10 层



§ 4. 利用函数实现指定的功能

4. 3. 函数的递归调用

4. 3. 3. 如何写递归函数

4. 3. 4. 如何读递归函数

- ★ 每次递归调用时，借助**栈**来记录调用的层次
- ★ 栈初始为空，每次递归函数被调用时在栈中增加一项，递归函数运行结束后栈中减少一项
- ★ 本次调用结束后，返回上次的调用位置，继续执行后续的语句
- ★ 重复操作至栈空为止

例1: 写出程序的运行结果及程序的功能

```
long fac(int n)
{
    if (n==0 || n==1)
        return 1;
    else
        return fac(n-1)*n;
}

int main()
{
    cout << "fac(5)=" << fac(5);
    return 0;
}
```

例1: 写出程序的运行结果及程序的功能

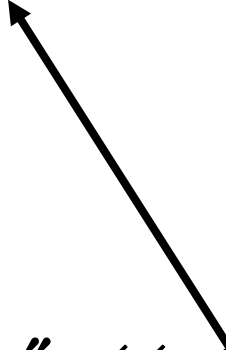
```
long fac(int n)
{
    if (n==0 || n==1)
        return 1;
    else
        return fac(n-1)*n;
}

int main()
{
    cout << "fac(5)=" << fac(5)
    return 0;
}
```

例1：写出程序的运行结果及程序的功能

```
long fac(int n)
{
    if (n==0 || n==1)
        return 1;
    else
        return fac(n-1)*n;
}

int main()
{
    cout << "fac(5)=" << fac(5);
    return 0;
}
```



fac(4)	5
fac(5)	

例1: 写出程序的运行结果及程序的功能

```
long fac(int n)
{
    if (n==0 || n==1)
        return 1;
    else
        return fac(n-1)*n;
}

int main()
{
    cout << "fac(5)=" << fac(5);
    return 0;
}
```

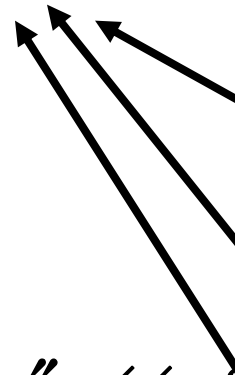
fac(3)	4
fac(4)	5
fac(5)	

例1: 写出程序的运行结果及程序的功能

```
long fac(int n)
{
    if (n==0 || n==1)
        return 1;
    else
        return fac(n-1)*n;
}

int main()
{
    cout << "fac(5)=" << fac(5) << endl;
    return 0;
}
```

fac(2)	3
fac(3)	4
fac(4)	5
fac(5)	



例1：写出程序的运行结果及程序的功能

```
long fac(int n)
{
    if (n==0 || n==1)
        return 1;
    else
        return fac(n-1)*n;
}

int main()
{
    cout << "fac(5)=" << fac(5) << endl;
    return 0;
}
```

fac(1)	2
fac(2)	3
fac(3)	4
fac(4)	5
fac(5)	

例1: 写出程序的运行结果及程序的功能

```
long fac(int n)
{  if (n==0 || n==1)
```

```
    return 1;
```

```
    else
```

```
    return fac(n-1)*n;
```

```
}
```

```
int main()
```

```
{
```

```
    cout << "fac(5)=" << fac(5);
```

```
    return 0;
```

```
}
```

fac(1)	2	1
fac(2)	3	
fac(3)	4	
fac(4)	5	
fac(5)		

例1：写出程序的运行结果及程序的功能

```
long fac(int n)
{   if (n==0 || n==1)
        return 1;
```

```
    else
```

```
        return fac(n-1)*n;
```

```
}
```

```
int main()
```

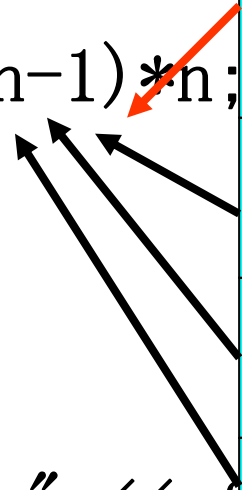
```
{
```

```
    cout << "fac(5)=" << fac(5);
```

```
    return 0;
```

```
}
```

fac(2)	3	2
fac(3)	4	
fac(4)	5	
fac(5)		



例1: 写出程序的运行结果及程序的功能

```
long fac(int n)
{
    if (n==0 || n==1)
        return 1;
    else
        return fac(n-1)*n;
}

int main()
{
    cout << "fac(5)=" << fac(5) << endl;
    return 0;
}
```

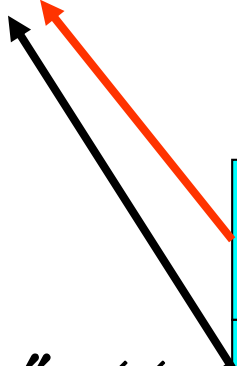
fac(3)	4	6
fac(4)	5	
fac(5)		



例1： 写出程序的运行结果及程序的功能

```
long fac(int n)
{
    if (n==0 || n==1)
        return 1;
    else
        return fac(n-1)*n;
}

int main()
{
    cout << "fac(5)=" << fac(5);
    return 0;
}
```

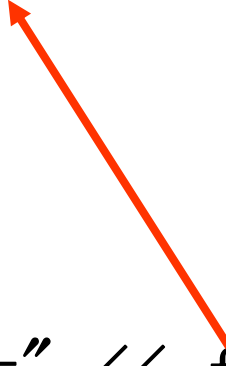


fac(4)	5	24
fac(5)		

例1: 写出程序的运行结果及程序的功能

```
long fac(int n)
{
    if (n==0 || n==1)
        return 1;
    else
        return fac(n-1)*n;
}

int main()
{
    cout << "fac(5)=" << f
    return 0;
}
```



fac(5)		120
--------	--	-----

例1: 写出程序的运行结果及程序的功能

```
long fac(int n)
{
    if (n==0 || n==1)
        return 1;
    else
        return fac(n-1)*n;
}

int main()
{
    cout << "fac(5)=" << fac(5);
    return 0;
}
```

fac(5)=120

fac(1)	2	1
fac(2)	3	2
fac(3)	4	6
fac(4)	5	24
fac(5)		120

例2：写出程序的运行结果

```
void f(int n, char ch)
{
    if (n==0)
        return;
    if (n>1)
        f(n-2, ch);
    else
        f(n+1, ch);
    cout << char(ch+n);
}

int main()
{
    f(7, 'k'); //VS2019中main无return不报错
}
```

例2: 写出程序的运行结果

```
void f(int n, char ch)
{
    if (n==0)
        return;
    if (n>1)
        f(n-2, ch);
    else
        f(n+1, ch);
    cout << char(ch+n);
}

int main()
{
    f(7, 'k');
}
```

7, k

例2：写出程序的运行结果

```
void f(int n, char ch)
{
    if (n==0)
        return;
    if (n>1)
        f(n-2, ch);
    else
        f(n+1, ch);
    cout << char(ch+n);
}

int main()
{
    f(7, 'k');
}
```

5, k

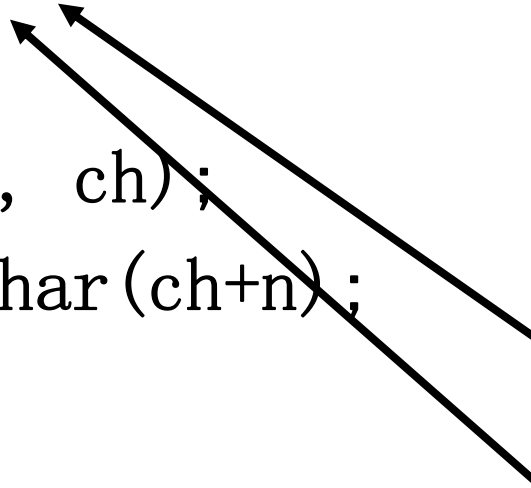
7, k

例2：写出程序的运行结果

```
void f(int n, char ch)
{
    if (n==0)
        return;
    if (n>1)
        f(n-2, ch);
    else
        f(n+1, ch);
    cout << char(ch+n);
}

int main()
{
    f(7, 'k');
}
```

3, k
5, k
7, k

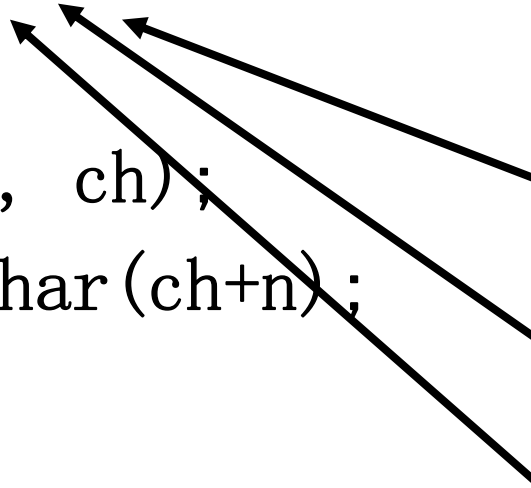


例2：写出程序的运行结果

```
void f(int n, char ch)
{
    if (n==0)
        return;
    if (n>1)
        f(n-2, ch);
    else
        f(n+1, ch);
    cout << char(ch+n);
}

int main()
{
    f(7, 'k');
}
```

1, k
3, k
5, k
7, k

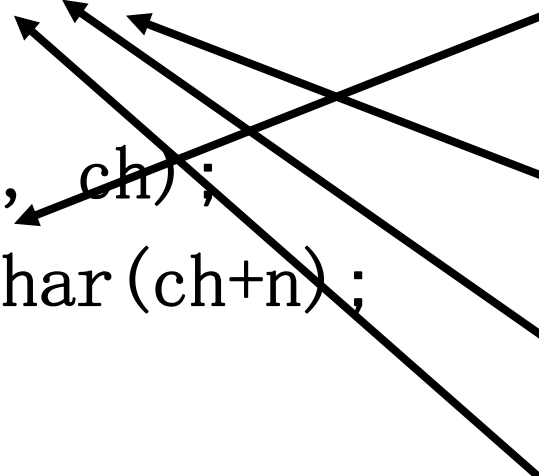


例2：写出程序的运行结果

```
void f(int n, char ch)
{
    if (n==0)
        return;
    if (n>1)
        f(n-2, ch);
    else
        f(n+1, ch);
    cout << char(ch+n);
}

int main()
{
    f(7, 'k');
}
```

2, k
1, k
3, k
5, k
7, k



例2：写出程序的运行结果

```
void f(int n, char ch)
```

```
{   if (n==0)
```

```
    return;
```

```
    if (n>1)
```

```
        f(n-2, ch);
```

```
    else
```

```
        f(n+1, ch);
```

```
    cout << char(ch+n);
```

```
}
```

```
int main()
```

```
{   f(7, 'k');
```

```
}
```

0, k

2, k

1, k

3, k

5, k

7, k

例2：写出程序的运行结果

```
void f(int n, char ch)
```

```
{  if (n==0)
```

```
    return;
```

```
    if (n>1)
```

```
        f(n-2, ch);
```

```
    else
```

```
        f(n+1, ch);
```

```
    cout << char(ch+n);
```

```
}
```

```
int main()
```

```
{  f(7, 'k');
```

```
}
```

0, k

2, k

1, k

3, k

5, k

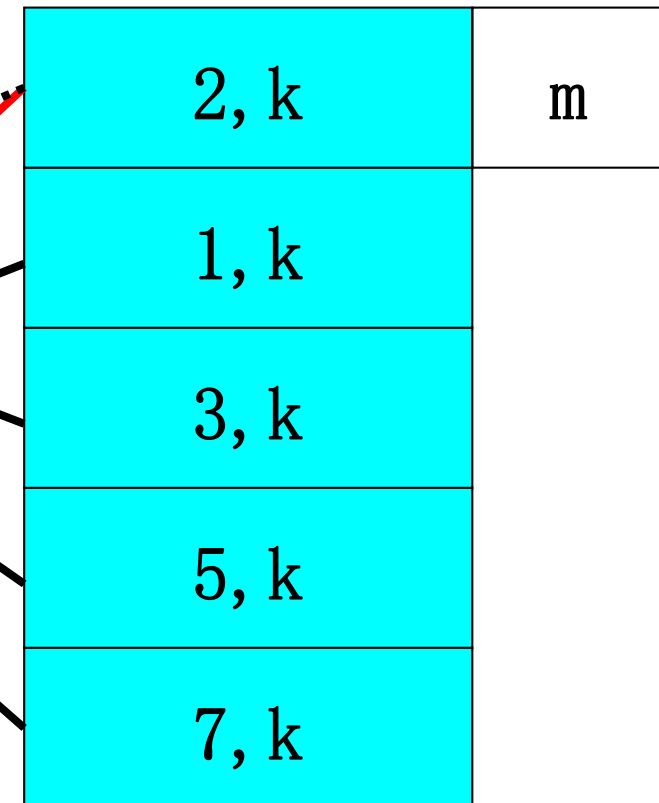
7, k

例2: 写出程序的运行结果

```
void f(int n, char ch)
{
    if (n==0)
        return;
    if (n>1)
        f(n-2, ch);
    else
        f(n+1, ch);
    cout << char(ch+n);
}

int main()
{
    f(7, 'k');
}
```

黑虚:上次保存现场位置
红实:本次恢复现场位置

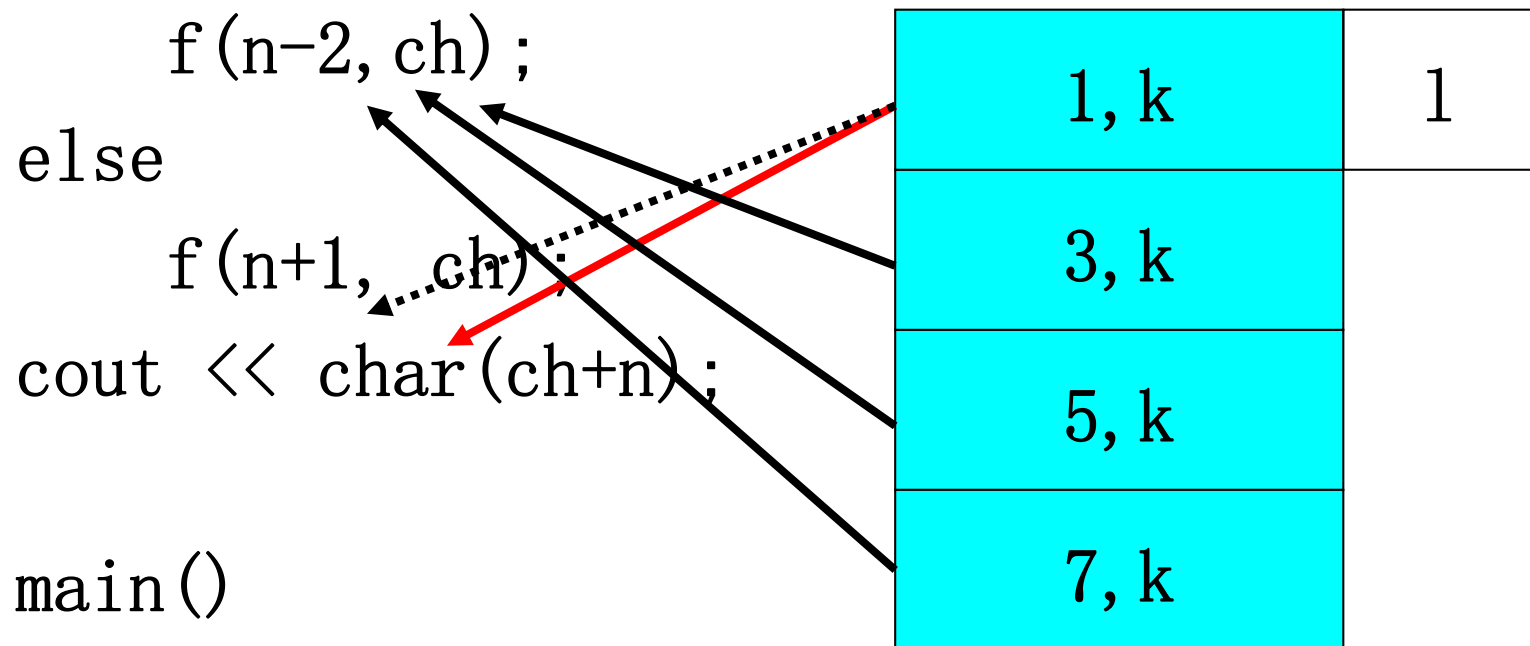


例2: 写出程序的运行结果

```
void f(int n, char ch)
{
    if (n==0)
        return;
    if (n>1)
        f(n-2, ch);
    else
        f(n+1, ch);
    cout << char(ch+n);
}

int main()
{
    f(7, 'k');
}
```

黑虚:上次保存现场位置
红实:本次恢复现场位置

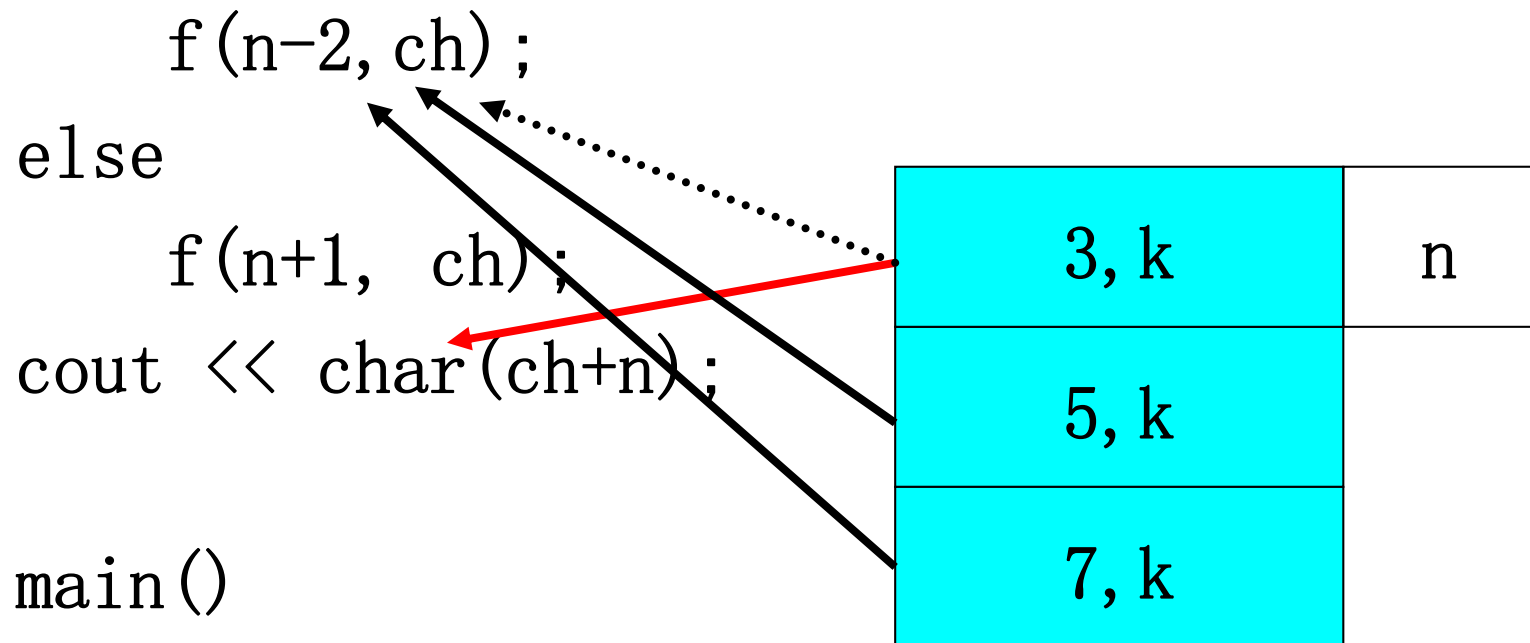


例2: 写出程序的运行结果

```
void f(int n, char ch)
{
    if (n==0)
        return;
    if (n>1)
        f(n-2, ch);
    else
        f(n+1, ch);
    cout << char(ch+n);
}

int main()
{
    f(7, 'k');
}
```

黑虚:上次保存现场位置
红实:本次恢复现场位置

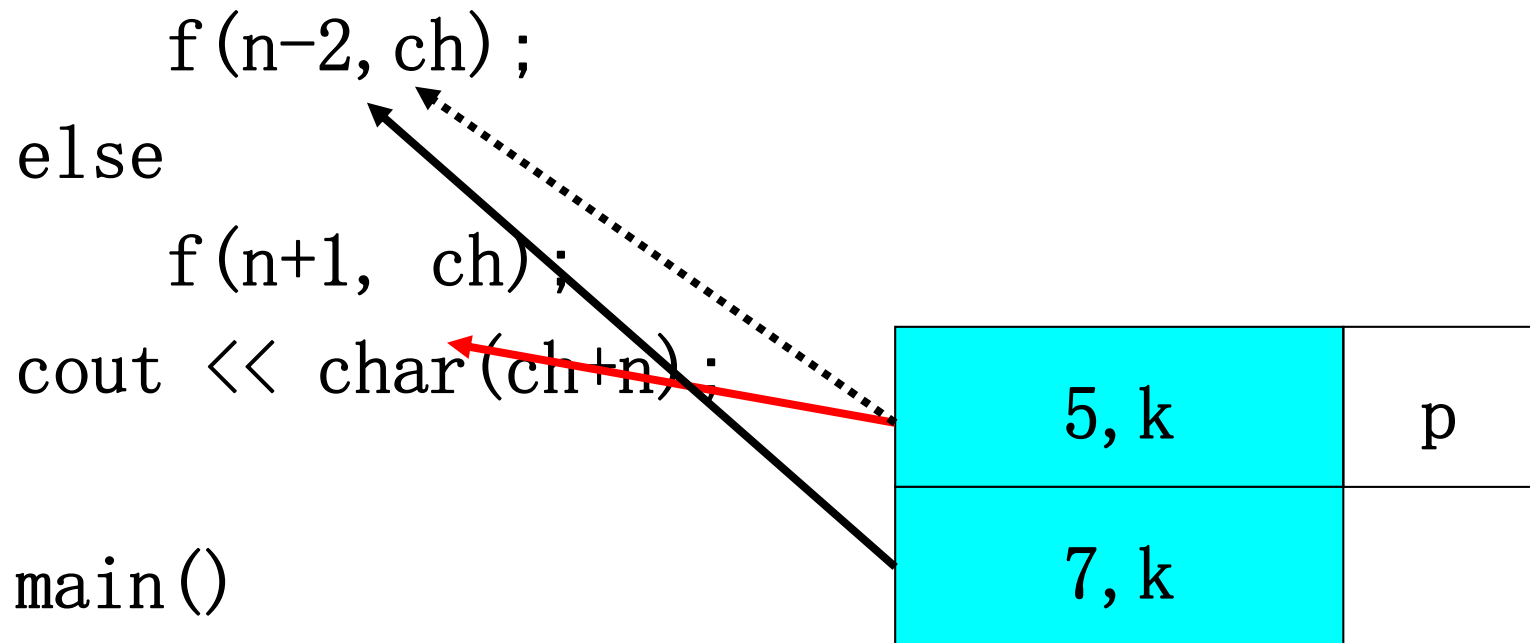


例2: 写出程序的运行结果

```
void f(int n, char ch)
{
    if (n==0)
        return;
    if (n>1)
        f(n-2, ch);
    else
        f(n+1, ch);
    cout << char(ch+n);
}

int main()
{
    f(7, 'k');
}
```

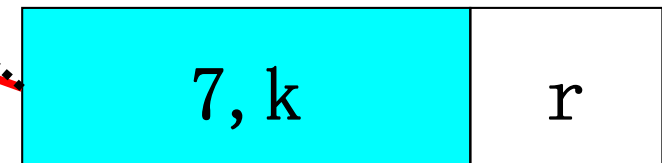
黑虚:上次保存现场位置
红实:本次恢复现场位置



例2：写出程序的运行结果

```
void f(int n, char ch)
{
    if (n==0)
        return;
    if (n>1)
        f(n-2, ch);
    else
        f(n+1, ch);
    cout << char(ch+n);
}

int main()
{
    f(7, 'k');
}
```



例2: 写出程序的运行结果

```
void f(int n, char ch)
{
    if (n==0)
        return;
    if (n>1)
        f(n-2, ch);
    else
        f(n+1, ch);
    cout << char(ch+n);
}

int main()
{
    f(7, 'k');
}
```

mlnpr

0, k
2, k
1, k
3, k
5, k
7, k

例3：写出程序的运行结果及功能

```
void f(int n, int k)
{
    if (n>=k)
        f(n/k, k);
    cout << n%k;
}

int main()
{
    f(14, 2); 1110
    cout << endl;
    f(65, 8); 101
    return 0;
}
```

请用栈的方式
自行画图理解

§ 4. 利用函数实现指定的功能

4.3. 函数的递归调用

4.3.5. 不设定终止条件的递归函数(错误的用法)

```
#include <iostream>
using namespace std;
int num = 0; //全局变量, 后面详述
void fun()
{   num ++; //用于统计fun被调用了多少次
    if (num % 1000 == 0)
        cout << "num=" << num << endl;
    fun();
}
int main()
{   fun();
    return 0;
}
```

双译器/四种模式, 观察结果
VS2019 : x86 / x64
Dev C++ : 32bit / 64bit

1、为什么会运行崩溃?

答:

2、不定义变量、定义10个int、10个double的情况下崩溃时打印的num值不同, 为什么?

答:

3、有兴趣自行研究各编译器如何改变堆栈大小

```
#include <iostream>
using namespace std;
int num = 0; //全局变量, 后面详述
void fun()
{   int a,b,c,d,e,f,g,h,i,j;
    a=b=c=d=e=f=g=h=i=j=10;
    num ++; //用于统计fun被调用了多少次
    if (num % 1000 == 0)
        cout << "num=" << num << endl;
    fun();
}
int main()
{   fun();
    return 0;
}
```

双译器/四种模式, 观察结果
VS2019 : x86 / x64
Dev C++ : 32bit / 64bit

```
#include <iostream>
using namespace std;
int num = 0; //全局变量, 后面详述
void fun()
{   double a,b,c,d,e,f,g,h,i,j;
    a=b=c=d=e=f=g=h=i=j=10;
    num ++; //用于统计fun被调用了多少次
    if (num % 1000 == 0)
        cout << "num=" << num << endl;
    fun();
}
int main()
{   fun();
    return 0;
}
```

双译器/四种模式, 观察结果
VS2019 : x86 / x64
Dev C++ : 32bit / 64bit