计算机科学基础I

——函数II

东南大学 集成电路学院 朱彬武

E-mail: bwzhu@seu.edu.cn

再次声明

- 期末机试（40%）
- 期末笔试（50%）
- 平时作业（10%）

可以用AI大模型学习如何编程，禁止用它替你完成编程

作业占比不高，但很多都是参考历年题目出的，希望大家好好消化，上传的参考答案下载下来检查一下错的地方

跳出循环——goto


```
    int total = 14;
    int two, five, seven;
    int exit = 0;
    for(two = 0; two <= total/2; two++){
        for(five = 0; five <= total/5; five++){
            for(seven = 0; seven <= total/7; seven++){
                if(two*2 + five*5 + seven*7 == total){
                    cout << "2角=" << two
                        << ", 5角=" << five
                        << ", 7角=" << seven << endl;
                    goto out;
                }
            }
        }
    }
out:
    return 0;
```

return提前结束函数


```
void solution(int total){
    for(int two = 0; two <= total/2; two++){
        for(int five = 0; five <= total/5; five++){
            for(int seven = 0; seven <= total/7; seven++){
                if(two*2 + five*5 + seven*7 == total){
                    cout << "2角=" << two
                        << ", 5角=" << five
                        << ", 7角=" << seven << endl;
                    return;
                }
            }
        }
    }
}
```

return代替break/goto，同样可以实现找到满足条件的组合，提前结束循环

return本身不能返回多个值

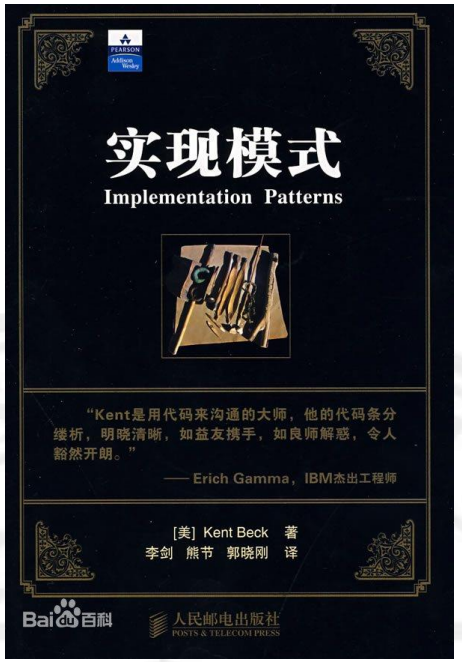

```
int solution(int total){
    for(int two = 0; two <= total/2; two++){
        for(int five = 0; five <= total/5; five++){
            for(int seven = 0; seven <= total/7; seven++){
                if(two*2 + five*5 + seven*7 == total){
                    cout << "2角=" << two
                        << ", 5角=" << five
                        << ", 7角=" << seven << endl;
                    return two, five, seven;
                }
            }
        }
    }
}

int main(){
    int total = 14;
    int two, five, seven;
    two, five, seven = solution(total);

}
```

逗号表达式

什么东西适合写成函数

函数适合什么场景

- 重复出现的代码：比如求阶乘：`int factorial(int n)`
- 有独立意义的逻辑：比如求三角形面积的函数：
  `double areaOfTriangle(int a, int b, int c)`
- 复杂或容易出错的代码：比如闰年判断：`bool isLeap(int year)`
- 需要参数化或可配置的代码：比如计算不同税率的个税：
  `double tax(int income, double rate)`
- ……



特别注意：我们倾向于编写简短，凝练的函数。一个函数应只做一件事，且把这件事做好。函数是代码的“乐高积木”——小而精的积木才能搭出灵活的结构。

函数注释

- 函数的简短说明
- 参数名以及参数的意义
- 返回值的意义
- 可选：注意事项、特殊情况、边界条件等

```
bool isPrime(int x);
/**
 * @brief 判断一个整数是否为质数
 *
 * @param x 待判断的整数
 * @return true  是质数
 * @return false 不是质数
**/
```

函数原型

函数先后关系


```
int pow(int base, int exponent){
    int res = 1;
    for(int i = 1; i <= exponent; i++){
        res *= base;
    }
    return res;
}

int main(){
    int res1 = pow(2, 7);
    cout << res1 << endl;

    int res2 = pow(5, 6);
    cout << res2 << endl;

    int res3 = pow(7, 5);
    cout << res3 << endl;
    return 0;
}
```

为什么要把pow函数写在main函数上面？明明程序是先从main函数开始执行

- 这是因为，在编译的时候，编译器要自上而下分析代码
- 在看到pow(2, 7)的时候，它需要知道pow函数的样子
- 也就是说pow()要几个参数，每个参数的类型如何，返回什么类型

如果调转顺序


```
int pow(int base, int exponent);
int main(){
    int res1 = pow(2, 7);
    cout << res1 << endl;

    int res2 = pow(5, 6);
    cout << res2 << endl;

    int res3 = pow(7, 5);
    cout << res3 << endl;

    return 0;
}

int pow(int base, int exponent){
    int res = 1;
    for(int i = 1; i <= exponent; i++){
        res *= base;
    }
    return res;
}
```

- 直接调转顺序更符合我们的阅读习惯，但是无法通过编译
- 需要在main函数前面加上一行函数原型：函数头加上分号“；”

函数原型（函数声明）


```
int pow(int base, int exponent);

int main(){
    int res1 = pow(2, 7);
    cout << res1 << endl;

    int res2 = pow(5, 6);
    cout << res2 << endl;

    int res3 = pow(7, 5);
    cout << res3 << endl;

    return 0;
}

int pow(int base, int exponent){
    int res = 1;
    for(int i = 1; i <= exponent; i++){
        res *= base;
    }
    return res;
}
```

```
int pow(int base, int exponent);
```

- 原型里可以不写参数的名字，甚至可以和函数头的名字不一样，但是为了方便阅读，最好写上且保持一致。

```
int pow(int, int);
int pow(int a, int b);
```

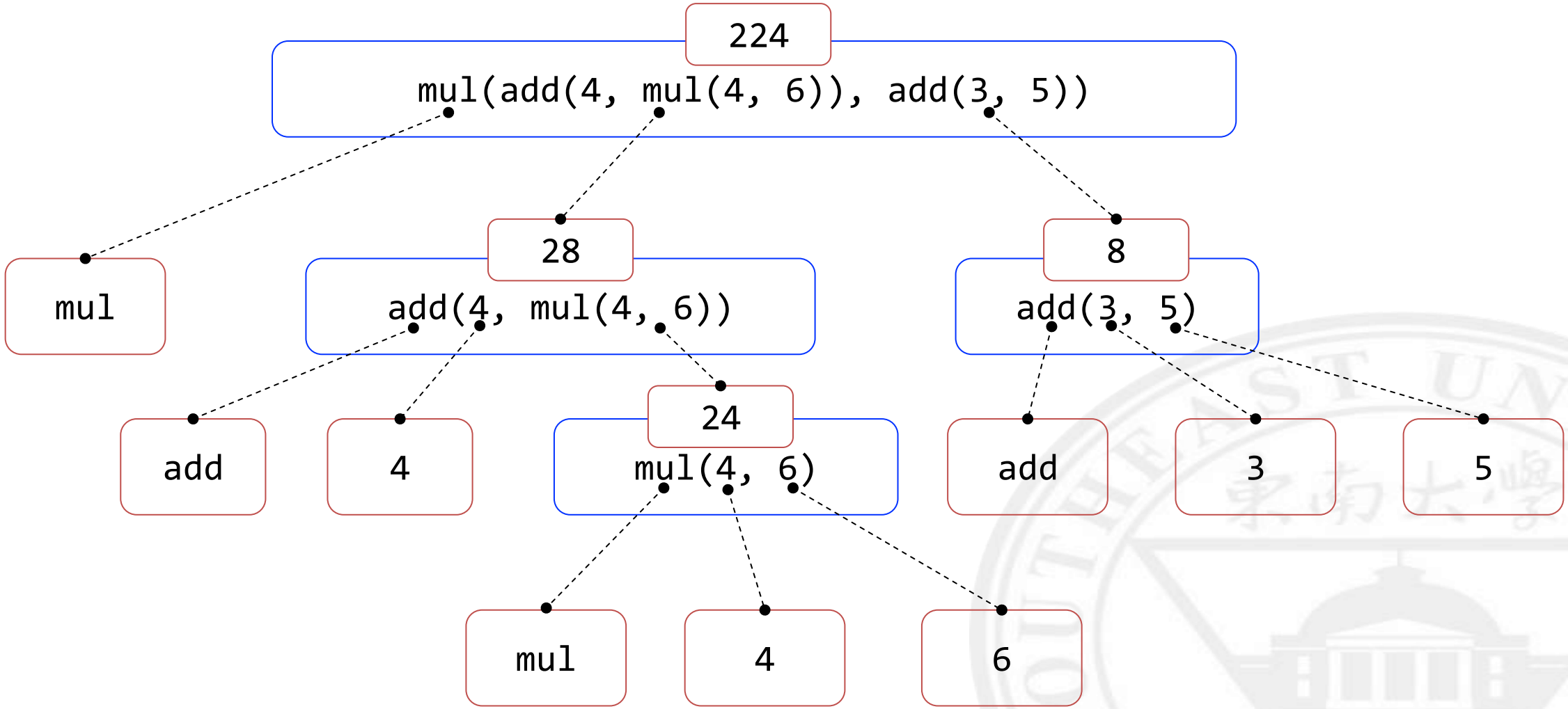
参数传递

参数传递

- 可以传递给函数的值是包括：
  - 字面量
  - 变量
  - 表达式：函数调用也是一种表达式

```
int a = 2, b = 3;
pow(2, 3);
pow(a, b);
pow(pow(2, 1), a + 1);
pow(pow(pow(2, 1), 1), 1 + 2)
```

嵌套的函数表达式


224
mul(add(4, mul(4, 6)), add(3, 5))
mul
28
add(4, mul(4, 6))
8
add(3, 5)
add
4
24
mul(4, 6)
add
3
5
mul
4
6

传值


```
void swap(int a, int b);

int main(){
    int a = 5;
    int b = 6;

    swap(a, b);

    cout << a << b << endl;
    return 0;
}

void swap(int a, int b){
    int t = a;
    a = b;
    b = t;
}
```

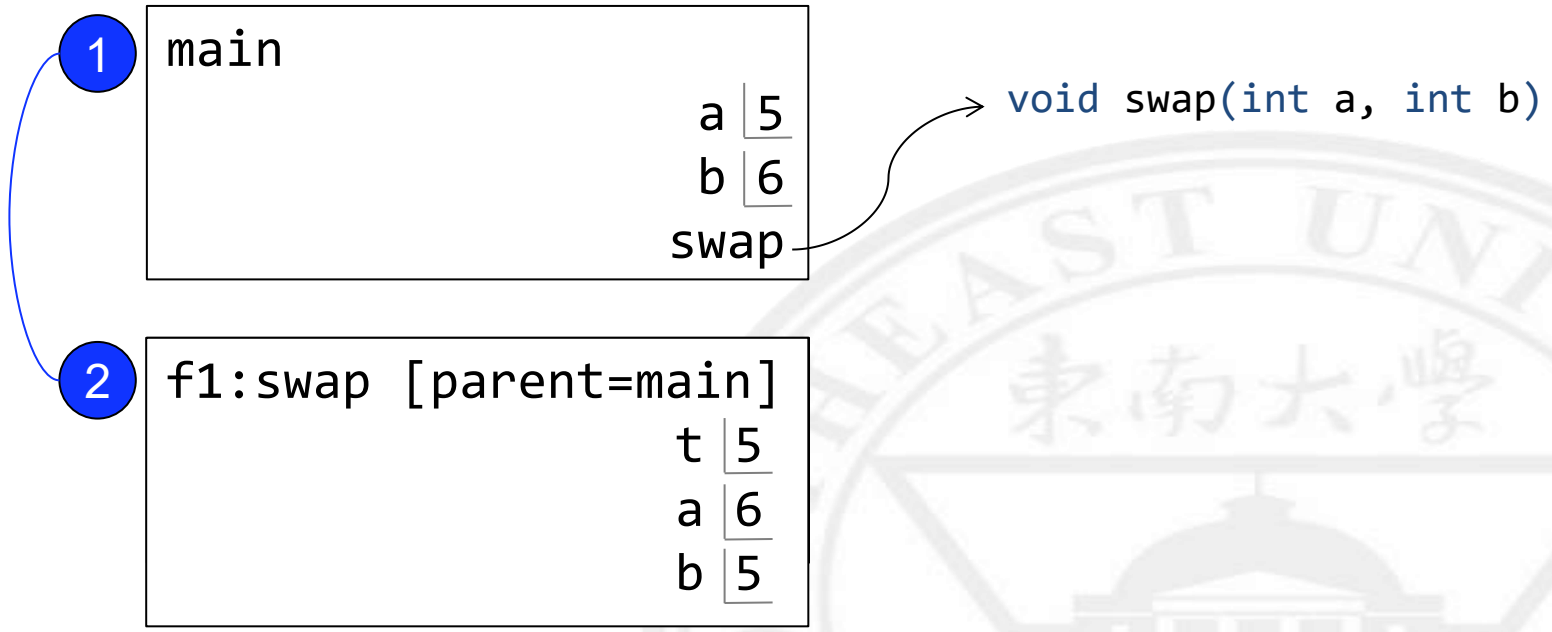
这样的代码能交换a和b的值吗？ 不能！

- 在调用swap函数的时候，我们只是把main函数中a和b的值交给了swap里的a和b
- swap函数里的a、b与main函数里的a、b完全没有任何关系，即使同名，但处在不同地方，仅仅是在调用的时候传递了值

函数调用过程示意图


```
void swap(int a, int b);

int main(){
    int a = 5;
    int b = 6;

    swap(a, b);

    cout << a << b << endl;
    return 0;
}

void swap(int a, int b){
    int t = a;
    a = b;
    b = t;
}
```

每个函数有自己的变量空间，参数也位于这个独立的空间中，和其他函数没有任何关系



局部变量

局部（本地）变量

- 函数的每次运行，就产生了一个独立的变量空间（栈帧），在这个空间中的变量，是函数的这次运行所独有的，称作局部变量

```
main
                    a 5
                    b 6
                  swap
```

```
f1:swap [parent=main]
                    t 5
                    a 6
                    b 5
```

- 定义在函数内部的变量就是局部变量
- 参数也是局部变量

变量的生存期和作用域

- 生存期：什么时候这个变量开始出现了，到什么时候它消亡了。
- 作用域：在（代码的）什么范围内可以访问这个变量（这个变量可以起作用）
- 对于局部变量，这两个问题的答案是统一的：大括号内

变量的生存期和作用域实例


```
double celToFah(double cel);

int main(){
    double fah, cel = 100;
    const double OFFSET = 32.0;
    const double SCALE = 9.0 / 5.0;
    fah = celToFah(cel);
}

double celToFah(double cel){
    double fah = cel * SCALE + OFFSET;
    return fah;
}
```

celToFah函数里能访问到常量SCALE和OFFSET吗？ 不能

局部变量的规则（1）

- 局部变量是定义在块内的
  - 它可以是定义在函数的块内
  - 也可以定义在语句的块内

- 程序运行进入这个块之前，其中的变量不存在，离开这个块，其中的变量就消失了

```
//----------------if语句------------------
int a = 5;
int b = 6;

if(a < b){
    int i = 10;
}
cout << i << endl;
```

即使去掉{}也访问不了i

```
//----------------for语句------------------
for(int i = 0; i < 2; i++){
    cout << i << endl;
}
cout << i << endl;

int i;
for(i = 0; i < 2; i++){
    cout << i << endl;
}
cout << i << endl;
```

注意两者区别

局部变量的规则（2）

- 局部变量还可以定义在任意一对大括号内
- 块外面定义的变量在里面仍然有效

```
int main(){
    int a = 2;
    {
        int b = 1;
        cout << a << endl;
    }
    cout << b << endl;  //error, 'b' was not declared in this scope
}
```

局部变量的规则（3）

- 块里面定义了和外面同名的变量则掩盖外面的，但是不能在一个块内定义同名的变量

```
int main(){
    int a = 2;
    {
        int a = 1;
        cout << a << endl; //1
    }
    cout << a << endl;  //2
}
```

函数递归

一个大家都听过的故事

从前有座山，山里有座庙，庙里有个老和尚，老和尚在给小和尚讲故事，他说：

从前有座山，山里有座庙，庙里有个老和尚，老和尚在给小和尚讲故事，他说：

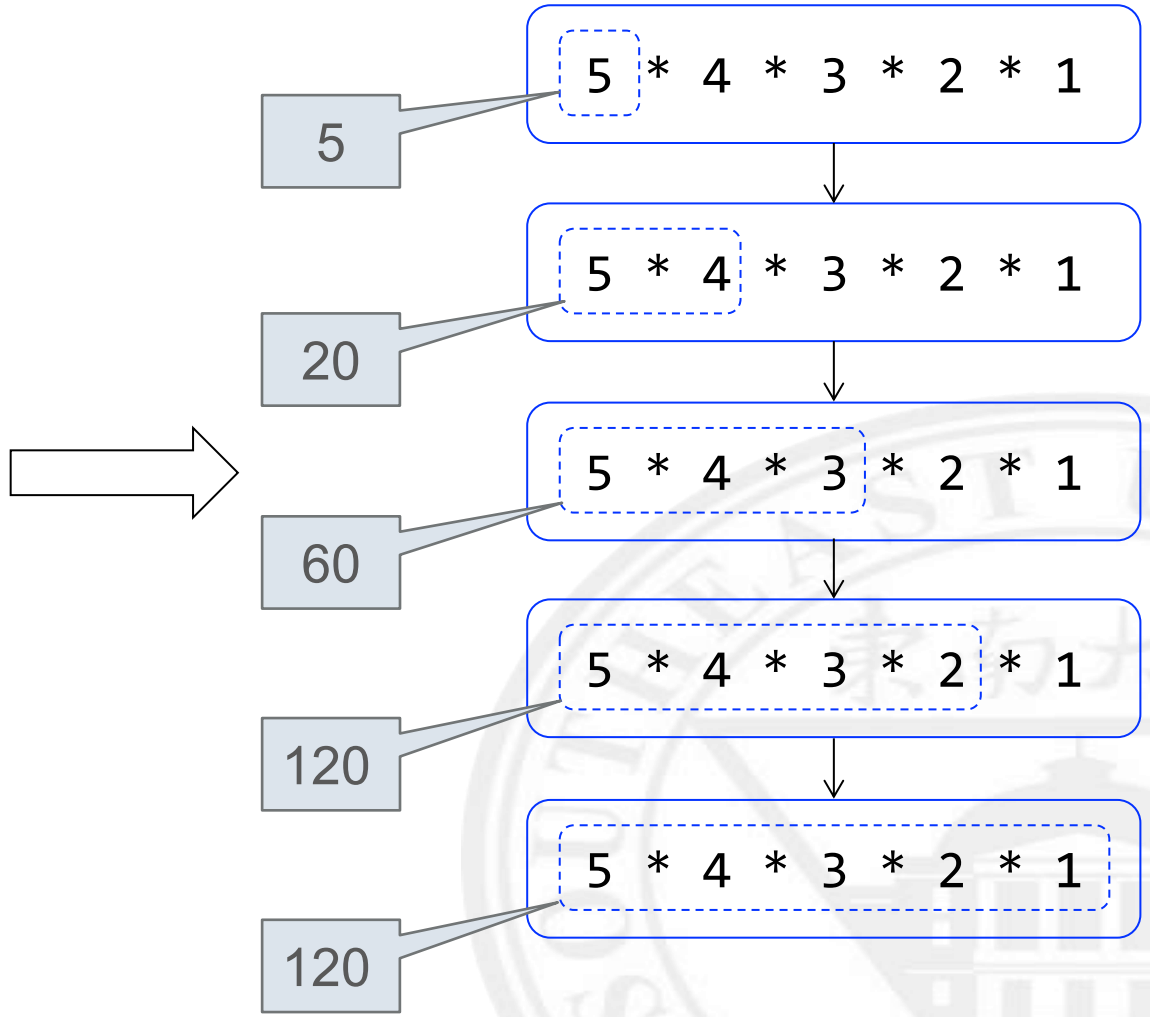
……

- 这就是递归，因为它不断地调用自己
- 但这不是我们想要的递归，因为它没有尽头，无限重复。除非在讲到比如第100次的时候加上一个结尾：故事讲到这里就结束了。

```
void tellStory() {
    cout << "从前有座山，山里有座庙，
            庙里有个老和尚，正在给小和尚讲故事。";
    cout << "他说\n";

    // 递归调用
    tellStory();   // 不停调用自己
}
```

阶乘——循环实现


```
int factorial(int n) {
    int result = 1;
    while(n >= 1){
        result *= n;
        n--;
    }
    return result;
}
```



阶乘——递归实现1.0

这么写会出现什么问题？ → 没有终止条件

```
int factorial(int n) {
    return n * factorial(n - 1);
}
```

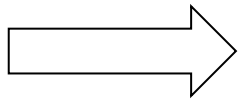
→

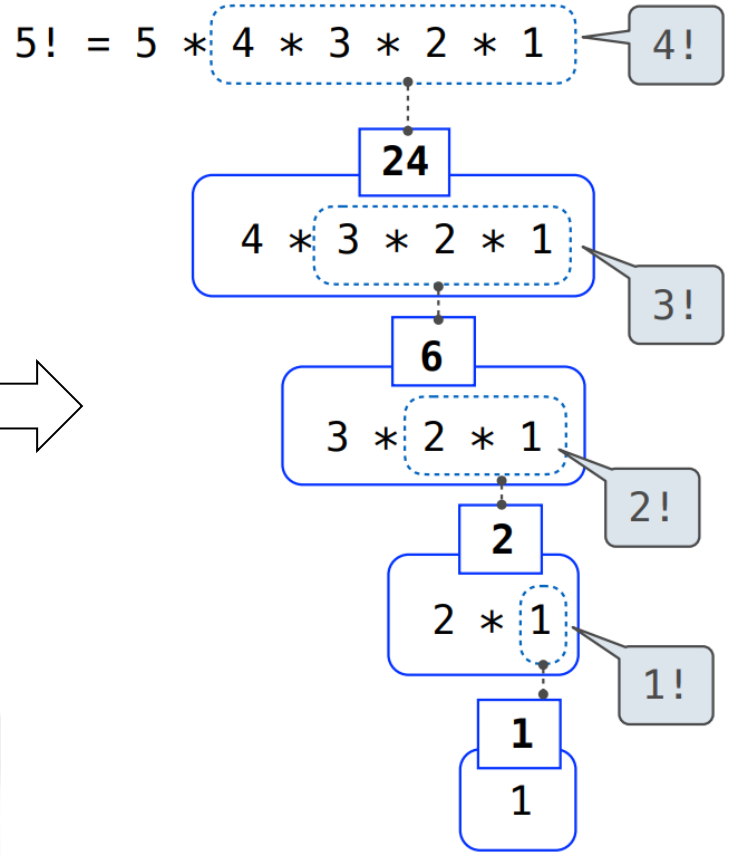
```
factorial(5)
 → factorial(4)
   → factorial(3)
     → factorial(2)
       → factorial(1)
         → factorial(0)
           → factorial(-1)
             → factorial(-2)
               ...
```

阶乘——递归实现

必须有：递归出口（终止条件）

```
int factorial(int n) {
    if (n == 1 || n == 0)  // 终止条件
        return 1;
    return n * factorial(n - 1);  // 递归调用
}
```





1. 判断是否满足终止条件
2. 若不满足：调用自身处理更小规模的问题
3. 返回并合并结果

斐波那契数列

就是斐波那契数列由0和1开始，之后的斐波那契数就是由之前的两数相加而得出。首几个斐波那契数是：1、1、2、3、5、8、……

```
int fibonacci(int n) {

    int prev = 0;
    int curr = 1;
    int nextValue = 1;

    for (int i = 3; i <= n; i++) {
        nextValue = prev + curr;
        prev = curr;
        curr = nextValue;
    }

    return nextValue;
}
```

斐波那契数列——递归实现

递推公式：　$F_0 = 0, F_1 = 1, F_n = F_{n-1} + F_{n-2}(n \geq 2)$

```
int fibonacci(int n) {
    // 递归终止条件
    if(...)
        ...
    return fibonacci(n - 1) + fibonacci(n - 2);  // 递归调用
}
```

留给大家上机课补全