

C语言程序设计





第四章 函数

4. 1库函数

4. 2自定义函数

4. 3函数的参数传递与返回值

4. 4递归函数

4. 5局部变量、全局变量与变量作用域

4. 6函数综合应用





学习目标：

- 理解函数在程序设计中的作用和地位
- 掌握函数的定义、原型声明和调用方法
- 熟练掌握函数的参数传递与返回值
- 理解局部变量、全局变量、静态变量和变量的作用域
- 掌握递归函数的编写技术
- 理解和掌握结构设计和模块化程序设计方法
- 了解地址和指针的概念，了解地址做函数参数的特点和作用





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

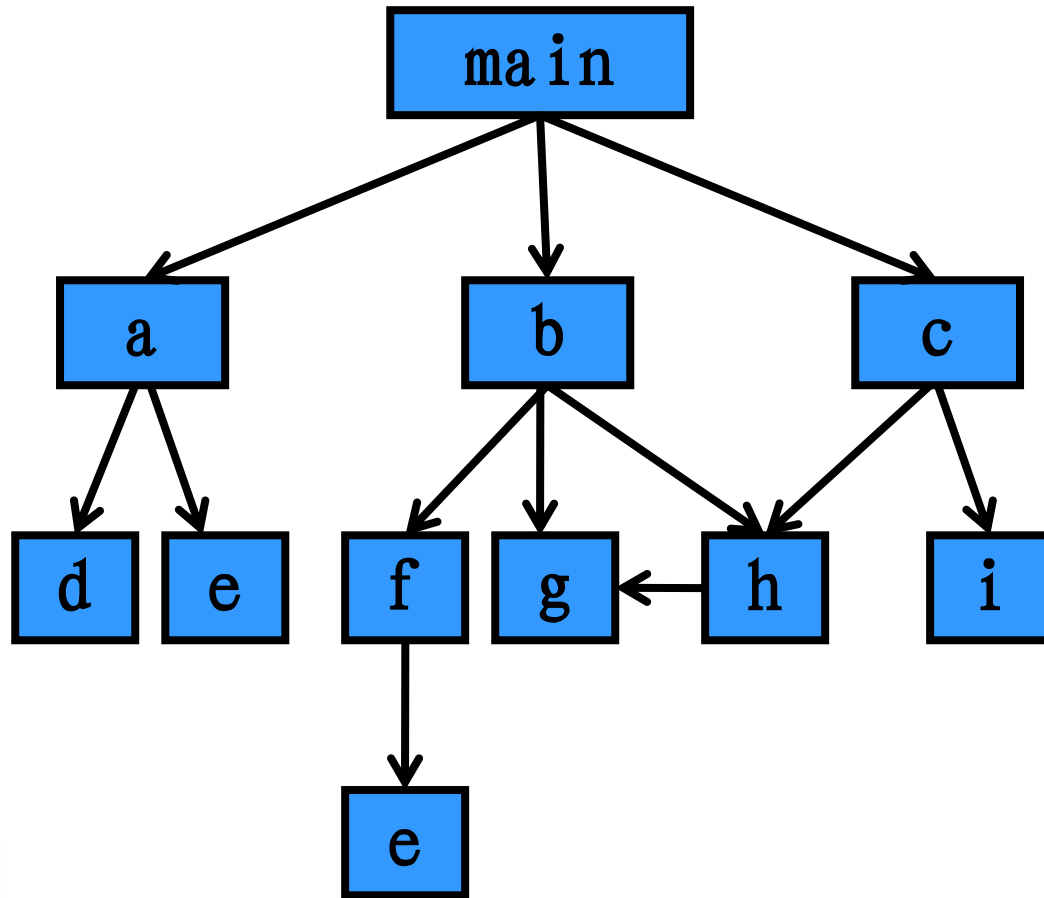




➤ 解决的方法：用模块化程序设计的思路

- 采用“**组装**”的办法简化程序设计的过程
- **可以使用库函数**：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
- **可以使用自己编写的函数**：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，函数的名字应反映其代表的功能
- 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
- 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







4. 1库函数

4. 1. 1导例：平方根表

4. 1. 2导例：随机生成一张扑克牌 （不讲）

4. 1. 3库函数使用方法

4. 1. 4常用的库函数





4.1.1 导例：平方根表

1. 问题描述

输出100以内整数的平方根表，要求每行输出10个。





2. 问题分析

方案1： 用牛顿迭代算法，先给出一个猜测值，用连续逼近方法求出平方根；首先设要求解的数为 y ，它的平方根为 x ，令 $x=y$ ；然后我们进入一个循环，不断的令 $x=(x+y/x)/2$ ，就是令 x 等于 x 和 y/x 的平均值，这样迭代多次后就可以得到 y 的平方根 x 的近似值，迭代次数越多，结果越精确。程序片段如下：

```
x=y;
```

```
for(int i=0;i<1000;i++)
```

```
    x=(x+y/x)/2;
```

方案2： C 语言提供了一个库函数**`sqrt()`**，使用时直接调用该库函数就可以计算出平方根值。



3. 算法描述

将上述方案2的过程以算法的形式描述为：

- (1) 输入2个整数并分别保存于整型变量 m ， n 中。
- (2) 输出平方根表时，要注意格式。首先输出表头，输出0~9的整数。
- (3) 然后按照格式要求输出平方根表， m 为行数0~9， n 为列数0~9，则表内对应第 m 行第 n 列的表值为 $\text{sqrt}(m*10+n)$ ，每行10个。





4. 程序实现

```
#include <stdio.h>
#include <math.h>      //数学类头文件
int main()
{   int m,n;
    for (n=0;n<10;n++)
        printf("%7d", n);      //输出表头
    printf("\n");
    for(m=0;m<10;m++)
    {   printf("%d",m);
        for (n=0;n<10;n++)
            printf("%7.4f" ,sqrt(m*10+n));  //调用库函数sqrt()
        printf("\n");    }
    return 0;
}
```





6. 程序分析

- 库函数`sqrt()`原型放在文件`math.h`文件中，所在若在程序中使用该函数，就需要在程序头加上语句`#include <math.h>`。
- 程序中语句段`for (n=0;n<10;n++)`是输出表头，输出列号：0~9；表头表示出0-9列的列号。
- 用输出函数`printf(“%7.4f”)`控制输出的值为保留4位小数的实数。





函数的定义



- ◆ 函数是指完成一个特定工作的独立程序模块。
 - ◆ 库函数：由C语言系统提供定义
如scanf () 、printf () 等函数
 - ◆ 自定义函数：需要用户自己定义
如计算圆柱体体积函数cylinder ()
- ◆ main()也是一个函数，C程序由一个main()或多个函数构成。
- ◆ 程序中一旦调用了某个函数，该函数就会完成特定的计算，然后返回到调用它的地方。
- ◆ 函数经过运算，得到一个明确的运算结果，并需要回送该结果。





4. 1. 3库函数使用方法

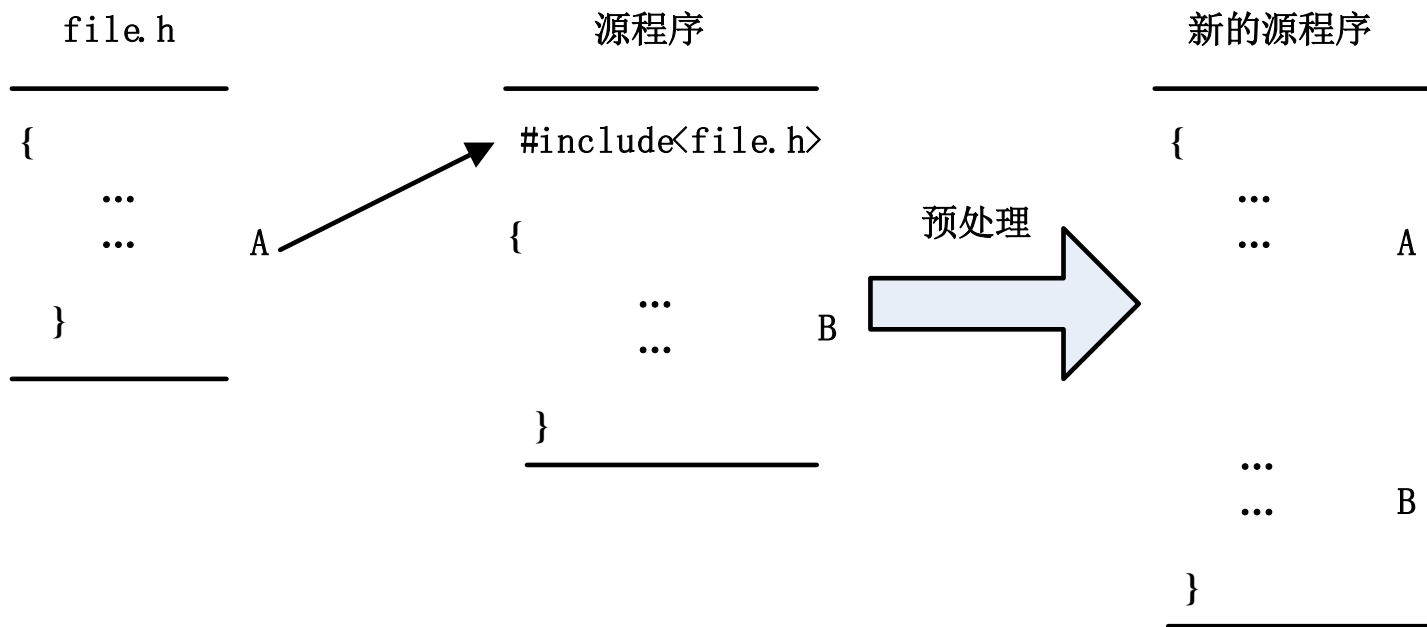
库函数的定义已经在C语言提供的标准函数库中，所以调用时，需要在程序的前面利用编译预处理命令**include**将相应的函数原型加入到程序中。

#include指令的一般形式如下：

#include<文件名> 或 #include “文件名”

例如： **#include<math.h>**







4. 1. 4常用的库函数

常用的函数头文件，各自包含的函数类别如下：

math.h: 包含与数学相关的函数

ctype.h : 包含与字符处理有关的函数

string.h : 包含与字符串处理有关的函数

stdio.h : 包含与输入输出有关的函数

stdlib.h : 包含与动态分配存储空间和数值转换有关的函数

process.h: 包含与过程控制有关的函数





double sin(double x)
double cos(double x)
double tan(double x)
double exp(double x)
double log(double x)
double pow(double x , double y)
double sqrt(double x)
double floor(double x)
double ceil(double x)





4. 2自定义函数

4. 2. 1导例：阶乘累加和

4. 2. 2导例：三色球问题（不讲）

4. 2. 3函数的定义、声明和调用

4. 2. 4函数调用过程分析





导例：计算圆柱体积



输入圆柱体的高和半径，求圆柱体积，

$$\text{volume} = \pi * r^2 * h。$$

要求定义和调用函数 **cylinder (r, h)** 计算圆柱体的体积。





```
/* 计算圆柱体积 */
#include <stdio.h>
int main( void )
{
    double height, radius, volume;
    double cylinder (double r, double h); /* 函数声明*/

    printf ("Enter radius and height: ");
    scanf ("%lf%lf", &radius, &height);
    /* 调用函数，返回值赋给volume */
    volume = cylinder (radius, height );
    printf ("Volume = %.3f\n", volume);

    return 0;
}
```



Enter radius and height: **3.0 10**

Volume = 282.743

/* 定义求圆柱体积的函数 */

double cylinder (double r, double h)

{

double result;

result = 3.1415926 * r * r * h; /* 计算体积 */

return result; /* 返回结果 */

}





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

```
int main( void )
```

```
{  double height, radius, volume;
```

```
    double cylinder (double r, double h);    /* 函数声明 */
```

```
    printf ("Enter radius and height: ");
```

```
    scanf ("%lf%lf", &radius, &height);
```

```
    volume = cylinder (radius, height );
```

```
    printf ("Volume = %.3f\n", volume);
```

```
    return 0;
```

```
}
```

```
double cylinder (double r, double h)
```

```
{  double result;
```

```
    result = 3.1415926 * r * r * h;
```

```
    return result; }
```

Enter radius and height: **3.0 10**

Volume = 282.743

问题：

函数是如何运行的？



4.2.1 导例：阶乘累加和



1. 问题描述

从键盘输入1个整数，计算1~n的各个数的阶乘的累加和，即 $1+2!+3!+\cdots+n!$ 。

2. 问题分析

实现各个数的阶乘的累加和，需要先计算每个数的阶乘，然后累加到一起。





方案1： 可用一个for语句循环实现。

```
result=1;sum=0;  
for ( i = 1; i <= n; i++ )  
{    result =result* i;  
    sum+=result; }
```

方案2： 可用一个函数factorial()专门来计算每个数的阶乘，然后用一个循环语句来计算1 ~ n的各个阶乘的累加的结果。





3. 算法描述

将上述方案2的过程以算法的形式描述为：

- ◆ 定义一个函数factorial(), 用于计算某个整数的阶乘。
- ◆ 输入1个整数并保存于整型变量n中。
- ◆ 将累加和变量sum初始化为0。
- ◆ 从1开始, 进行n次循环处理。
 - 调用factorial(), 分别以1~n的各个数字为实参;
 - 将factorial()返回结果与sum值累加。
- ◆ 输出sum值。





4. 程序实现

```
#include <stdio.h>
double factorial (int i); //函数声明
int main(void)
{
    int i, n;
    double sum=0;
    printf("Enter 1 integers:");
    scanf("%d",&n);
    for(i = 1; i <=n; i++ )
        sum = sum + factorial (i); //函数调用
    printf("1!+...+%d! = %.0f\n", n,sum);
    return 0;
}
```

```
double factorial (int i)
//函数定义
{
    int j;
    double result = 1;
    for (j = 1; j<= i;j++)
        result = result *j ;
    return result ;
//返回结果
}
```



4. 2. 3函数的定义、声明和调用



一、函数的定义

函数定义的基本形式是：

```
函数类型 函数名（形参表）    /*函数头，没有分号*/  
{  
    函数实现过程    /*函数体*/  
}
```

形参表 给出函数所有形参的名称和类型，它的格式为：

类型1 形参1，类型2 形参2，……类型n 形参n

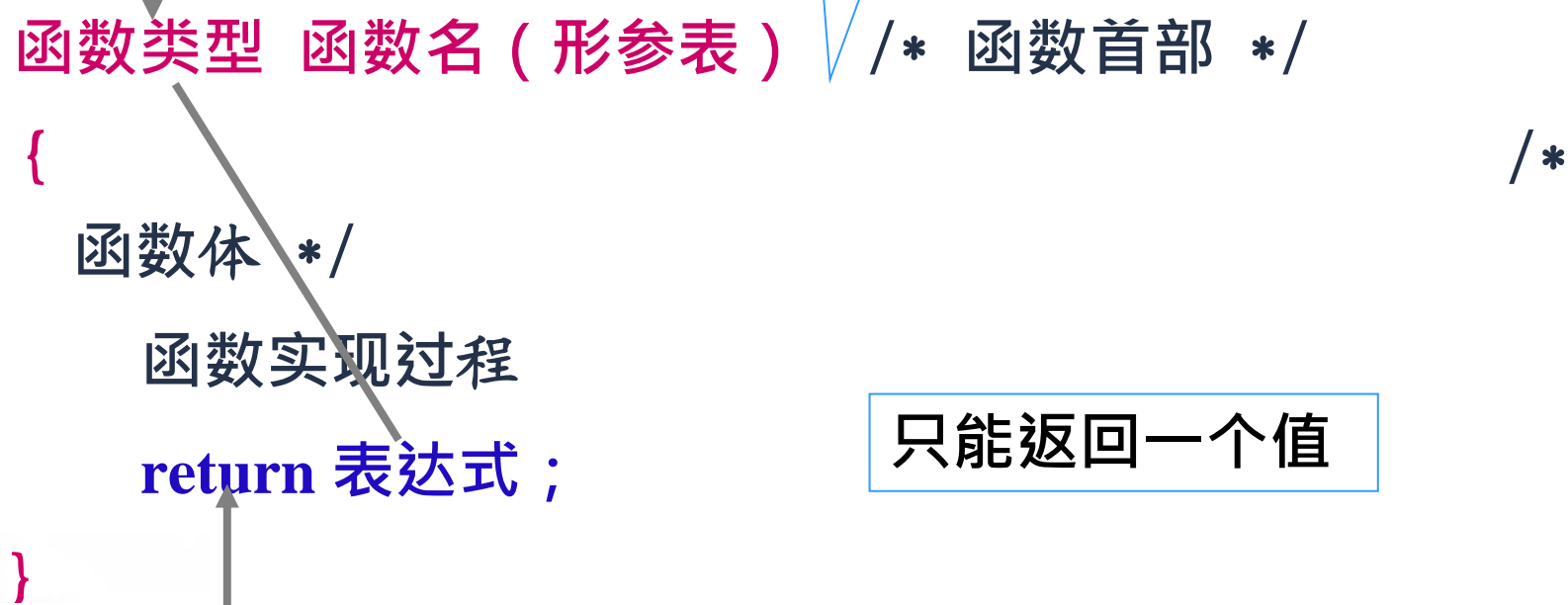




```
double cylinder (double r, double h)
{
    double result;
    result = 3.1415926 * r * r * h;
    return result;
}
```

函数返回值的类型

没有分号



只能返回一个值

把函数运算的结果回送给主函数



1、分析函数的定义



函数类型

函数名

形参表

double cylinder (double r, double h) /* 函数首部 */
{ /* 函数体，写在一对大括号内 */

double result;

result = 3.1415926 * r * r * h; /* 计算圆柱体积 */

return result; /* 返回运算结果 */

与函数类型一致





形参



不能写成 `double r, h`

`double cylinder (double r, double h)`
{ `double result`;
 `result = 3.1415926 * r * r * h`;
 `return result`;
}

函数类型 函数名 (形参表)

{
 函数实现过程
 `return 表达式` ;
}

类型1 参数1 ， 类型2 参数2 ， ， 类型n 参数n

参数之间用逗号分隔，每个参数前面的类型都必须分别写明





```
double factorial (int i)    //函数定义
{
    int j;double result = 1;
    for (j = 1; j<= i;j++)
        result = result *j ;
    return result ;    //返回结果
}
```

```
例 : int maxValue(int d1, int d2)
{
    if(d1>=d2)
        return d1;
    else
        return d2;
}
```



2、函数的调用



- 定义一个函数后，就可以通过程序来调用这个函数。
- 调用标准库函数时，在程序的最前面用`#include`命令包含相应的头文件。
- 调用自定义函数时，程序中必须有与调用函数相对应的函数定义。





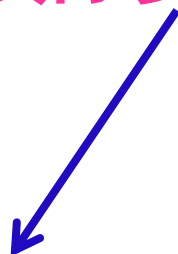
函数调用的形式



函数调用的一般形式为：

函数名（实际参数表）；

`factorial (i);`



实际参数与形式参数的数据类型和个数一一对应，以分号结束





函数的调用



例 求两个数中的最大值

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

```
void main( )
```

```
{ int a,b,c,max(int x, int y);
```

```
  scanf("%d,%d",&a,&b);
```

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

```
  printf("Max is %d",c);
```

```
}
```

```
int max(int x, int y)
```

```
{ int z;
```

```
  if(x>y)z=x; else z=y;
```

```
  return(z);
```

```
}
```

实参

形参

定义函数时，形参
调用函数时，实参





函数调用的形式



对于实现计算功能的函数，函数调用通常出现在两种情况：

➤ 赋值语句

```
volume = cylinder(radius, height );
```

➤ 输出函数的实参

```
printf(“%f”, cylinder(radius, height ) );
```





函数调用的过程



- 计算机在执行程序时，从主函数main开始执行，如果遇到某个函数调用，主函数被暂停执行，转而执行相应的函数，该函数执行完后，将返回主函数，然后再从原先暂停的位置继续执行。
- 函数遇return返回主函数





分析函数调用的过程



```
#include <stdio.h>
int main( void )
{
    double height, radius, volume;
    double cylinder (double r, double h);
    printf ("Enter radius and height: ");
    scanf ("%lf%lf", &radius, &height);
    volume = cylinder (radius, height );
    printf ("Volume = %.3f\n", volume);
    return 0; }
```

```
double cylinder (double r, double h)
{
    double result;
    result = 3.1415926 * r * r * h;
    return result;
}
```

调用函数

实参→形参

执行函数中的语句

返回调用它的地方



参数传递



- 函数**定义**时的参数被称为**形式参数**（简称**形参**）
double cylinder (double **r**, double **h**) ;
- 函数**调用**时的参数被称为**实际参数**（简称**实参**）
volume = cylinder (**radius**, **height**);
- 参数传递：**实参**→**形参** **单向传递**
 - 在参数传递过程中，实参把值复制给形参。
 - 形参和实参**一一对应**：数量一致，类型一致，顺序一致
 - **形参**：变量，用于接受实参传递过来的值
 - **实参**：常量、变量或表达式





函数结果返回



- 完成确定的运算，将运算结果返回给主调函数。
- 函数结果返回的形式：
 - `return` 表达式；
 - `return` (表达式)；





函数原型声明



只写函数定义中的第1行（函数首部），并以分号结束。

函数类型 函数名(行参表);

double cylinder (double r, double h);

double factorial (int n);

void pyramid (int n);

- 函数必须先定义后调用，将主调函数放在被调函数的后面，就像变量先定义后使用一样。
- 如果自定义函数在主调函数的后面，就需要在函数调用前，加上函数原型声明。
- 函数声明：说明函数的类型和参数的情况，以保证程序编译时能判断对该函数的调用是否正确。





小结:

要调用函数，
必须先要声明！

在执行函数调用时，实参把值计算出来，拷贝
给相应位置的形参；函数执行完后，通过
`return()`，可返回一个结果值。

实参与形参
个数相同、类型一致

形参的改变
不影响实参
变量的值

只能返回一个结果，
类型与函数定义时一致



4.3 函数的参数传递与返回值

4.3.1 导例：爬动的蠕虫

4.3.2 导例：日K蜡烛图

4.3.3 函数的参数传递

4.3.4 函数的返回值





导例：求 π 的值



输入精度 e ，使用格里高利公式求 π 的近似值，精确到最后一项的绝对值小于 e 。要求定义和调用函数`funpi(e)` 求 π 的近似值。

$$\frac{\pi}{4} = 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \dots$$





源程序



/* 用格里高利公式计算 π 的近似值,
精度为e */

```
#include <stdio.h>
#include <math.h>
int main (void)
{ double e, pi;
  double funpi (double e);

  printf ("Enter e:");
  scanf ("%lf", &e);
  pi = funpi (e);
  printf ("pi = %f\n", pi);

  return 0;
}
```

Enter e: 0.0001

pi = 3.1418

```
double funpi (double e)
{ int denominator, flag;
  double item, sum;
  flag = 1;
  denominator = 1;
  item = 1.0;
  sum = 0;
  while (fabs (item) >= e){
    item = flag * 1.0 / denominator;
    sum = sum + item;
    flag = -flag;
    denominator = denominator + 2;
  }
  return sum * 4;
}
```





导例：判断素数的函数



求100以内的全部素数，每行输出10个。素数就是只能被1和自身整除的正整数，1不是素数，2是素数。

要求定义和调用函数prime (m)判断m是否为素数，当m为素数时返回1，否则返回0。

算法描述：对2~100之间的每个数进行判断，若是素数，则输出该数。

```
for(m = 2; m <= 100; m++)
```

```
    if (m是素数)
```

```
        prime(m) != 0
```

```
        printf("%d ", m);
```





例5-4 源程序

```
#include <stdio.h>
#include <math.h>
int main(void)
{   int count, m;
    int prime (int m);
    count = 0;
    for(m = 2; m <= 100; m++){
        if ( prime(m) != 0 ){
            printf("%6d", m );
            count++;
            if (count %10 == 0)
                printf ("\n");
        }
    }
    printf ("\n");
}
```

```
int prime (int m)
{   int i, n;
    if ( m == 1 ) return 0;
    n = sqrt (m);
    for( i = 2; i <= n; i++){
        if (m % i == 0){
            return 0;
        }
    }
    return 1;
}
```



输出

```
/* 输出数字金字塔
#include <stdio.h>
int main (void)
{
```

```
void pyramid (int n);
pyramid(5);
return 0;
```

```
}
```

```
void pyramid (int n)
{
```

```
int i, j;
for (i = 1; i <= n; i++){
    for (j = 1; j <= n-i; j++)
        printf(" ");
    for (j = 1; j <= i; j++)
        printf(" %d ", i);
    putchar ('\n');
```

```
}
}
```

```
for (i = 1; i <= n; i++) {
    for (j = 1; j <= n-i; j++)
        printf(" ");
    一行中的数字显示
}
```

```
for (i = 1; i <= n; i++) {
    一行中的空格处理 ;
    一行中的数字显示
}
```

```
/* 函数声明 */
```

```
/* 调用函数，输出数字金字塔 */
```

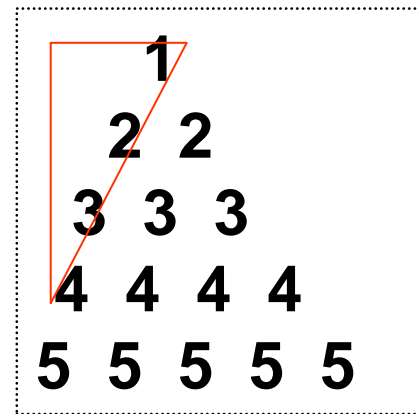
```
/* 函数定义 */
```

```
/* 需要输出的行数 */
```

```
/* 输出每行左边的空格 */
```

```
/* 输出每行的数字 */
```

```
/* 每个数字的前后各有一个空格 */
```





4.3.2导例：日K蜡烛图

1. 问题描述

从键盘输入股市开盘价格、最高价格、最低价格、收盘价格，输出当日蜡烛图名称。





2. 问题分析

- ◆ 股票价格涨跌趋势常用蜡烛图技术中的K线图来表示
- ◆ 日K 线为例：每天对应一根蜡烛小图 在一天对应的蜡烛图中，要表示4个价格：开盘价格、收盘价格、全天的最高价和最低价。
- ◆ 收盘价格<开盘价格，表示为“实心蓝白蜡烛”；
- ◆ 收盘价高>开盘价，表示为“空心红蜡烛”；
- ◆ 开盘价=收盘价，为“十字红蜡烛”。
- ◆ 最低价格<开盘价和收盘价，称为“有下影线”
- ◆ 最高价>收盘价和开盘价，称为“有上影线”。





3. 算法描述

将上述过程以算法的形式描述为：

- 输入4个整数并分别保存于实型变量open、high、low、close中，分别表示开盘价格、最高价格、最低价格、收盘价格。
- 设计两个函数candle()和shadow()，分别处理蜡烛图的两个特性是否实心、空心或是否有上、下影线。
- candle()函数：比较open 和close，若close<open，则输出“实心蓝白蜡烛”；若close>open，则输出“空心红蜡烛”；否则，直接输出“十字红蜡烛”。
- shadow()函数：low、high分别和open、close比较，若low同时小于close 和open，则输出“有下影线”；若high同时大于close和open，则输出“有上影线”，否则输出“无上影线和下影线”。





4. 程序实现

```
#include <stdio.h>
void candle(float open,float close);
void shadow(float open,float high,float low,float close);
main()
{
    float open,high,low,close;
    printf("Enter 4 real numbers:");
    scanf("%f%f%f%f",&open,&high,&low,&close);
    shadow(open,high,low,close);           //函数调用，参数传递
    candle(open,close);
}
```





```
void shadow(float open,float high,float low,float close)
```

```
{ if (low<open && low<close)
```

```
    printf("有下影线");
```

```
    if (high>open && high>close)
```

```
        printf("有上影线"); }
```

```
void candle(float open,float close)
```

```
{ if(close<open)
```

```
    printf("实心蓝白蜡烛");
```

```
    else if (close>open)
```

```
        printf("空心红蜡烛");
```

```
    else    printf("十字红蜡烛"); //函数无返回值 }
```





6. 程序分析

void candle(float open,float close)

void shadow(float open,float high,float low,float close)

有参数无返回值。void说明函数没有返回值。





不返回运算结果的函数定义



表示不返回结果

↓
void 函数名 (参数表) /* 函数首部 */
{ /* 函数体 */
 函数实现过程
 return ; /* 可以省略return */
}

这类函数通常用于屏幕输出等

不能省略, 否则
函数类型被默认定义为**int**



不返回运算结果的函数定义



- 由于函数没有返回结果，函数调用不可能出现在表达式中，通常以独立的调用语句方式，如`pyramid(5);`;
- 不返回结果的函数，在定义、调用、参数传递、函数声明上，思路完全与以前相同，只是函数类型变为`void`。
- 它适用把一些确定的、相对独立的程序功能包装成函数。
 - 主函数通过调用不同的函数，体现算法步骤
 - 各步骤的实现由相应函数完成
 - 简化主函数结构，以体现结构化程序设计思想。





4.3.3 函数的参数传递

在调用函数时，实参与形参结合的具体过程是：

- 计算实参表达式的值；
- 将实参的值按赋值转换规则转换成对应形参的数据类型；
- 为形参分配存储空间；
- 将类型转换后的实参的值传递给对应的形参变量，然后执行函数。





从函数是否有参数和返回值的角度，可以将函数分为：

①有参数有返回值函数

```
int max(int x, int y )  
{  
    if(x>y) return x;  
    else return y;  
}
```

②有参数无返回值函数

```
void delay(long t)  
{  
    for(int i=1;i<t; i++) ; //延迟一个小的时间片  
}
```





③ 无参数有返回值函数

```
int in( )  
{ int x;  
  scanf("%d",&x);  
  return x;  
}
```

④ 无参数无返回值函数

```
void hello( )  
{  
  printf("helloworld!\n");  
}
```





C语言中，函数的参数传递有两种方式，即传值和传地址。

传值：简单地将实参的值复制一份给形参，一旦复制完成，实参与其对应的形参便没有任何关系，这时在函数内对形参的任何改变都不会影响到实参。

传地址：将实参地址传递给形参，在函数内对形参所对应数据的处理实际上就是处理对应的实参。简单地理解就是此时实参和形参同一的，对形参的处理就是对对应的实参的处理。

C语言中，当函数的参数为指针类型或数组类型时 采用传地址的方式，其他类型采用传值方式。





4.3.4 函数的返回值

函数返回值有两种类型：

- ◆ 完成确定的运算，由运算结果返回给主调函数，称为**有返回值的函数**；
- ◆ 完成指定的工作，没有确定的运算结果需返回给主调函数，称为**无返回值的函数**。

通常用于实现结构化程序设计中的过程模块，
函数类型用void指定。





1、有返回值函数

函数执行结束后调用者返回一个**执行结果**，称为函数的返回值。有返回值函数在函数定义时必须说明返回值的**类型**，在函数体中由**return**语句给出具体的返回值。

有返回值return语句的一般形式如下：

return 表达式；

或

return (表达式)；

- 先求解表达式的值，再返回其值。通常表达式的**结果类型**与函数的**返回值类型**一致，如果两者不一致，以函数类型为准。
- return语句的作用有两个：一是结束函数的运行；二是带着运算结果（表达式的值）返回主调函数。



函数有返回值：函数执行完毕后，返回一个相应类型的数值。

用return语句返回该值：return 表达式；

如：

```
printf("The average is %6.2f", average(dataArray, n));
```

```
ave= average(dataArray, n);
```





2、无返回值函数

只完成某种特定的处理，函数执行后无须向调用者返回计算结果。

无返回值函数在函数定义时必须将返回值的类型说明为 **void**（即空类型），函数体中的 **return** 语句只结束函数的执行。如果函数类型为 **void**，则函数返回直接用 **return** 语句，不必跟上一个表达式。

无返回值 **return** 语句的一般形式如下：

return ;

在函数定义中也可以**没有** **return** 语句，此时函数执行到最后一条语句。





4. 4递归函数

4. 4. 1导例：假币问题（三分法）

4. 4. 2导例：Fibonacci数列

4. 4. 3递归函数的执行过程

4. 4. 4递归函数的效率分析





4.4.2 导例：假币问题（三分法）

1. 问题描述

有 n 个硬币，已知有一个是假币，而且它的重量比真币小，现在有一个天平，问最多需要称几次可以把那个假币找出来。






方案2：采用三分法，即将硬币分成三份



- 用 $f(n)$ 表示当硬币数为 n 时需要称重的次数。
- 当 $n=1$ 时，不用称即可知道：必有假币的话，只有一枚，即是假币；
- 当 $n=2$ 时，称1次即可知道：重量轻的的那个硬币是假的；
- 当 $n=3$ 时，选出两个用天平称1次即可：如果两个硬币相等，第三个硬币是假币；如果两个不等，那么重量轻的的那个硬币是假的；
- 当 $n>3$ 时，如果 n 能被3整除，则将硬币分成三份，同样按照前面的称重方法，公式为 $f(n)=1+f(n/3)$ 。如果 n 不能被3整除，那么将它先分成个数相同的两堆，第3堆个数比前两堆多一个或少一个，最多可能的个数为 $n/3+1$ 个，公式为 $f(n)=1+f(n/3+1)$ 。


$$f(n) = \begin{cases} 0 & n=1 \\ 1 & n=2 \\ 1 & n=3 \\ 1+f(n/3+1) & n>3 \end{cases}$$

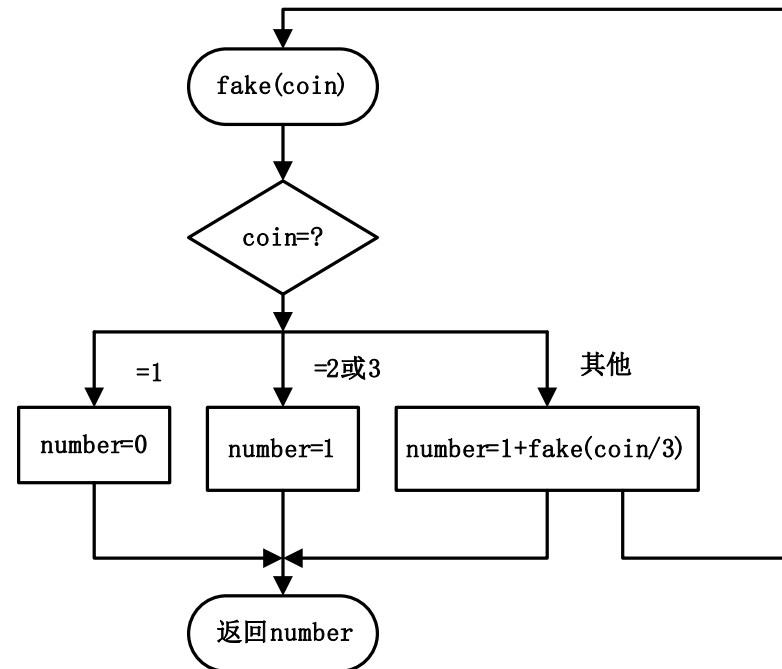
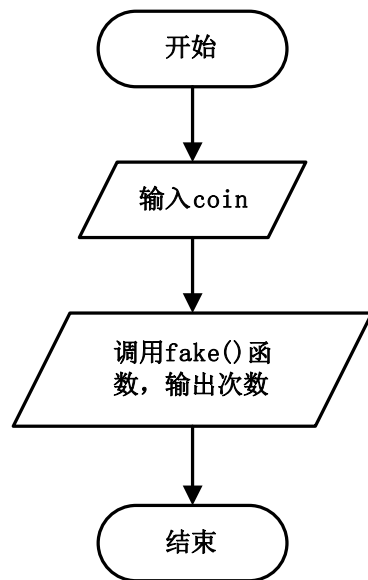


3. 算法描述

将上述方案2过程以算法的形式描述为：

- 定义一个整形变量coin，调用函数fake(coin)，计算给出的硬币数量通过多少次可以找出假币
- fake(coin)函数按照上述递归公式，设计找出假币的次数。
- 根据coin的值，用swtich 语句，判断coin分别等于1、2、3时需要的次数，大于3时使用递归，最后得出结果。







4. 程序实现

```
#include <stdio.h>
int fake(int coin);
main()
{ int coin;
  printf("输入硬币数:");
  scanf("%d",&coin);
  printf("需要称%d次 \n", fake(coin)); //递归函数调用 }
```

```
int fake(int coin)
{ int number;
  switch(coin)
  { case 1: number=0;break;
    case 2: case 3: number=1;break;
    default: number=1+fake(coin/3);}
  return number; }
```





6. 程序分析

(1) 递归函数的声明、定义和调用和普通自定义函数一样，只是在函数体实现过程中，会调用函数自己，但是参数值不一样

number=1+fake(coin/3) 参数不是**coin**,

递归函数一定要设计**递归出口**，也就是递归函数**终止递归**下去的条件，

case 1: number=0;break;

(2) 如果需要确定哪枚硬币是假币的话，可以给假币按顺序编号，用数组存储硬币编号，采用二分法确定哪一枚是假币。





4.4.2 导例：Fibonacci 数列

1. 问题描述

从键盘输入一个整数 n ，输出斐波那契数列 (fibonacci) 的前 n 项。要求每行打印5个数后换行。

2. 问题分析

斐波那契数列为 $1, 1, 2, 3, 5, 8, 13, \dots$ 即从第三项开始，各项值都等于前两项值之和。斐波那契数列的定义为：

$$f(n) = \begin{cases} 1 & n=0 \\ 1 & n=1 \\ f(n-1)+f(n-2) & n>1 \end{cases}$$



方案1：采用非递归方法，程序片段如下：

```
first=1,second=1;
scanf("%d",&n);
for(i=0; i<n;i++)
{
    result=first+second;
    printf("%.0f",result);
    first=second;
    second=result;
}
```

方案2：采用递归方法，定义递归函数fib(k)，获得斐波那契数列的第k个数。



3. 算法描述

将上述方案2的过程以算法的形式描述为：

1. 定义一个整形变量 **value**。

2. 调用函数 **fib(n)**，输出第 **n** 项值。

3. **fib(n)** 函数按照上述递归公式，实现递归出第 **n** 项的值。

当 **n=0** 或 **1** 时，表示计算 **fib(0)** 或 **fib(1)**，直接等于 **1**；

如果把计算第 **n** 项值写成函数 **fib(n)**，那么 **fib(n)** 的实现依赖于 **fib(n-1)** 和 **fib(n-2)**，以此类推。





4. 程序实现

```
#include <stdio.h>

int fib(int k);

main()
{ int n,value,count=0;
  printf("Enter 1 integers\n:");
  scanf("%d",&value);
  for(n=0;n<value;n++)
  { printf("%10d",fib(n));
    count++;
    if (count%5==0)
      printf("\n");  }
}
```





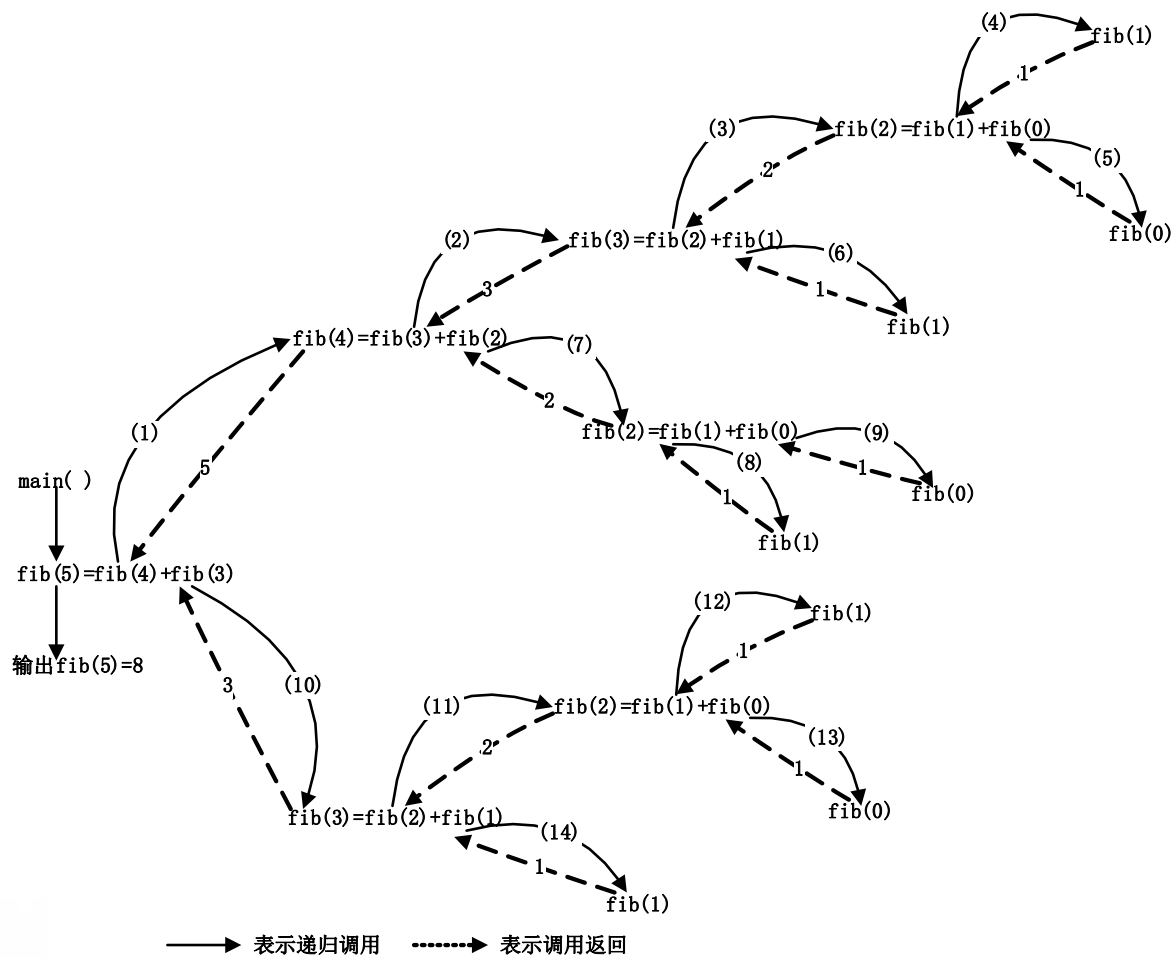
递归函数

```
int fib(int n)
{
    int f;
    if (n==0)
        f= 1;
    else if (n==1)
        f=1;
    else
        f=fib(n-1)+fib(n-2);
    return f;
}
```





6. 程序分析

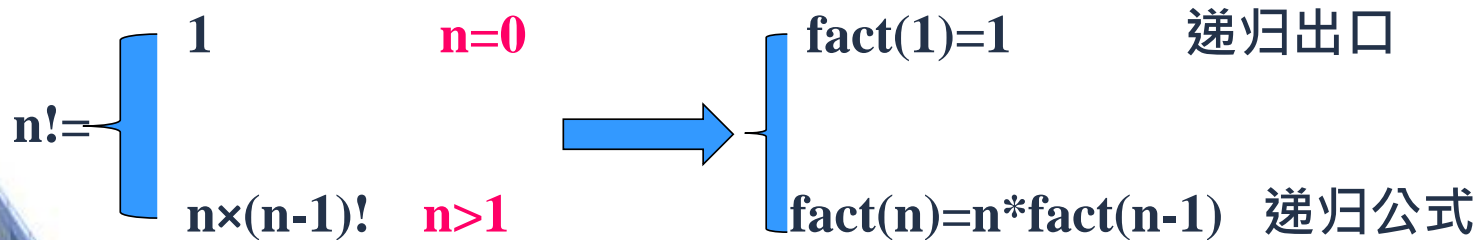




4. 4. 3递归函数的执行过程

递归函数要注意两点:

- ①**递归出口**: 递归的结束条件, 说明到何时不再递归调用 ; 否则无限制递归, 终将使栈空间溢出 ;
- ②**递归公式**: 当前函数结果与准备调用的函数结果之间的关系, 即原问题是如何分解为子问题的。



$$\sum_{i=1}^n i = n + \sum_{i=1}^{n-1} i$$



➤ 实现阶乘递归算法的递归函数

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

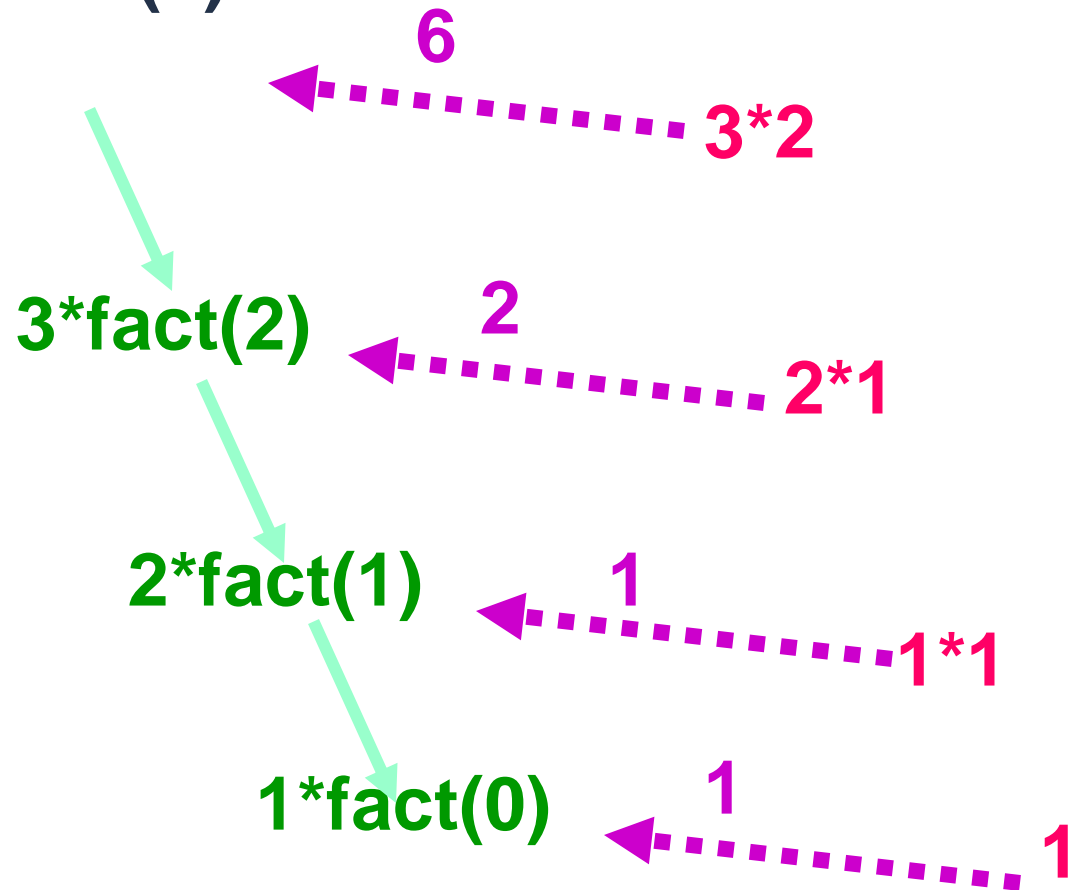




§ 递归函数的调用过程



$y = \text{fact}(3)$





$n!$ (另一种方法)



```
#include <stdio.h>
double fact(int n) ;
int main(void)
{ int n;
  scanf ("%d", &n);
  printf ("%f", fact (n) );
  return 0;
}
double fact(int n)
{ double result;
  if (n==1 || n == 0)
    result = 1;
  else
    result = n * fact(n-1);
  return result;
}
```

求 $n!$ 递归定义

$$n! = n * (n-1)! \quad (n > 1)$$

$$n! = 1 \quad (n = 0, 1)$$

递归出口

递归式

~~$fact(n) = n * fact(n-1);$~~



递归程序的内部执行过程

函数的递归调用类似于多个函数的嵌套调用，只不过调用函数和被调用函数是**同一个函数**。递归调用的内部执行过程如下：

- ①**运行开始时**，系统设立工作栈来保存每次调用的运行环境，包括形参、局部变量和返回地址；
- ②**递归调用前**，将调用函数的形参、局部变量以及调用后的返回地址进栈；
- ③**调用结束后**，将栈顶数据出栈，恢复调用前的运行环境，使相应的形参和局部变量恢复为调用前的值，然后从返回地址指定的位置继续执行调用函数。





4. 6函数综合应用

4. 6. 1导例：模拟银行ATM机存取款

4. 6. 2导例：贷款计算器

4. 6. 3程序主体框架的设计与实现

4. 6. 4模块化程序设计的基本特征





4.6.1 导例：模拟银行ATM机存取款

1 · 问题描述

模拟银行ATM机存取款，实现单个帐户的查询、存款、取款、转帐、修改密码等功能。





2 · 问题分析

进行**模块化设计**，上述各项需求：查询、存款、取款、转帐、修改密码分别用不同函数实现。由main()函数调用各函数来实现上述需求。首先输入密码进行判断，密码正确后输出一个简单菜单显示项，操作有(1查询，2存款，3取款，4转帐，修改密码，0退出)，读取用户的操作选择；再根据操作选择项，分别调用各自函数，经多次操作直到输入操作为0结束。





3. 算法描述

(1) 设全局变量balance保存存款余额, key保存密码

(2) 模块划分, 确定各函数的功能、参数和返回值。

menu()函数: 显示菜单选择项, 读取操作选择。

check()函数: 返回存款余额。

deposit()函数: 存款

withdraw()函数: 取款

transfer()函数: 转帐

changekey()函数: 修改密码





4.6.2 导例：贷款计算器

1 · 问题描述

贷款计算器是根据贷款情况计算还款情况的计算器。贷款计算器有很多种，这里设计一款简单的贷款计算器：选择等额本金或等额本息的还款方式时，计算每月的月供、利息总额和还款总额。





2. 问题分析

简易贷款计算器的功能如下：

- **录入**贷款信息，即输入贷款方式（等额本息、等额本金）、贷款总额、按揭年数、年利率、利率折扣等信息。
- **显示**贷款信息，即输出贷款方式（等额本息、等额本金）、贷款总额、按揭年数、年利率、利率折扣等信息。
- **计算**还款信息。即计算出每月的月供、利息总额、还款总额等信息。
- **退出**系统。





3. 算法描述

- (1) 设计菜单函数`menu()`，根据菜单项选择要调用的函数，设计`input()`、`show()`、`pay()`函数分别实现贷款信息输入、显示贷款信息和显示还款信息。
- (2) 定义全局变量，`pattern`表示还款方式，`loan`表示贷款总额，`year`表示按揭年数，`rate`表示年利率，`discount`表示利率折扣。





4. 6. 3程序主体框架的设计与实现

模块化（Modularization）是把系统分割成能完成独立功能的模块，明确规定各模块的输入输出规格，使模块的界面清楚，功能明确。

模块有以下基本属性：

- 名称：较好表达该模块的功能
- 接口：模块的输入和输出。
- 功能：模块实现的功能。
- 逻辑：模块内部如何实现功能及所需要的数据。
- 状态：模块的调用与被调用关系。

通常，模块从调用者那里获得输入数据，然后把产生的输出数据返回给调用者。



4. 6. 4模块化程序设计的基本特征

- **自顶向下：**程序设计时，应先考虑总体步骤，后考虑步骤的细节；先考虑全局目标，后考虑局部目标。先从最上层总目标开始设计，逐步使问题具体化。不要一开始就追求众多的细节。
- **逐步求精：**对于复杂的问题，其中大的操作步骤应该再将其分解为一些子步骤的序列，逐步明晰实现过程。
- **函数实现：**通过逐步求精，把程序要解决的全局目标分解为局部目标，再进一步分解为具体的小目标，把最终的小目标用函数来实现。问题的逐步分解关系，构成了函数间的调用关系。





在设计函数时要注意：

- **函数功能的设计**：结合模块独立性原则，函数的功能要单一，一个模块一个功能，不要设置多用途的函数，否则会降低模块的聚合度。对于多处使用的同一个计算或操作过程，应当将其设计成一个独立的函数，达到一处定义、多处使用的目的，避免功能间的重复。
- **函数规模设计**：函数的规模要小，尽量控制代码行数，使得函数更易于阅读、理解、调试和维护。
- **函数接口的设计**：结合模块独立性的原则，函数的接口包括函数的参数（入口）和返回值（出口），不要设计过于复杂的接口，合理选择、设置、控制参数的数量，尽量不要使用全局变量，否则会增加模块的耦合度。采用定义局部变量作为函数的临时工作单元，使用参数和返回值作为函数与外部进行数据交换的方式。只有当确实需要多个函数共享的数据时，才定义其为全局变量。





THE END

