模块化程序设计

第三章 模块化程序设计

- 3.1 模块化程序设计思想
- 3.2 函数定义
- 3.3 函数调用
- 3.4 函数的原型与声明
- 3.5 函数的嵌套与递归
- 3.6 库函数
- 3.7 变量的作用域与存储类型
- 3.8 指针与函数
- 3.9 典型例题

模块化程序设计思想

- 一个较大的程序一般应分解为若干个程序模块，每个模块用来实现一个特定的功能。这样可以把复杂的问题划分成若干小问题，通过对小问题的处理，最终解决复杂的大问题。
- 在C语言中，这些独立的模块就是由函数完成的。
- 在程序设计中，将一些常用的功能模块编写成函数，这样可以减少重复编写程序段的工作量。

模块化程序设计的特点

- 模块相对独立，功能单一
- 编写相对简单，可以独立编写调试
- 可集体开发，缩短开发周期。不同的模块可以由不同的人员开发，最终能够合成完整的程序
- 开发出的模块，可在不同的应用程序中多次使用，减少重复劳动，提高开发效率
- 测试、更新以模块为单位进行而不会影响其他模块

引例

计算：

$$\frac{k!}{m!+n!}$$

- 阶乘的实现：

```
fk = 1;
for (i=1;  i<=k;  i++)
{
    fk = fk * i;
}
```

```
#include  <stdio.h>
int  main(void)
{
    int  i, k, m, n;
    float  fk, fm, fn;
    scanf("%d%d%
    fk = 1;
    for (i=1; i<=k; i++)
        fk = fk * i;
    fm = 1;
    for (i=1; i<=m; i++)
        fm = fm * i;
    fn = 1;
    for (i=1; i<=n; i++)
        fn = fn * i;
    printf("%f\n", fk/(fm+fn));
    return 0;
}
```

如果换成如下形式怎么做？

$$\frac{\sqrt{k}}{\sqrt{m}+\sqrt{n}}$$

反复使用的代码段

```
#include <stdio.h>
t  factorial(int n)
nt  i;
oat  f;
 = 1;
for (i=1;  i<=n;  i++)
        f  = f  *  i;
return  f;
}
d)
 n;
float  fk, fm, fn;
scanf("%d%d%d", &k, &m, &n);
fk = factorial(k);
fm = factorial(m);
fn = factorial(n);
printf("%f\n", fk/(fm+fn));
return 0;
}
```

函数定义

```
float  factorial(int n)
{
    int i;
    float f = 1;
    for (i=1; i<=n; i++)
        f = f * i;
    return f;
}
```

定义格式：

函数类型 函数名(形式参数表)

{

函数体：{ 声明部分 / 执行部分

}

其中形式参数表为：

类型名 形参1,类型名 形参2,……

- 说明
  - 定义函数时，函数头的形参列表的右括号后不能有分号
  - 形参列表中每一个形参都应指定一种数据类型
  - 不能在函数体中再次定义形参列表中出现的变量

函数调用

- 函数定义的目的就是使用它，而函数的使用是通过函数调用实现的
- 一般形式：函数名(实际参数表)
- 说明
  - 实参的个数、类型、顺序应与被调函数定义时所要求的参数个数、类型、顺序一致
  - 各实参之间用逗号分隔，每个实参前不要再加数据类型
  - 实际参数表（简称实参）可以为空，但括号不能省略
  - 实参可以是常数、变量或表达式，必须有确定值

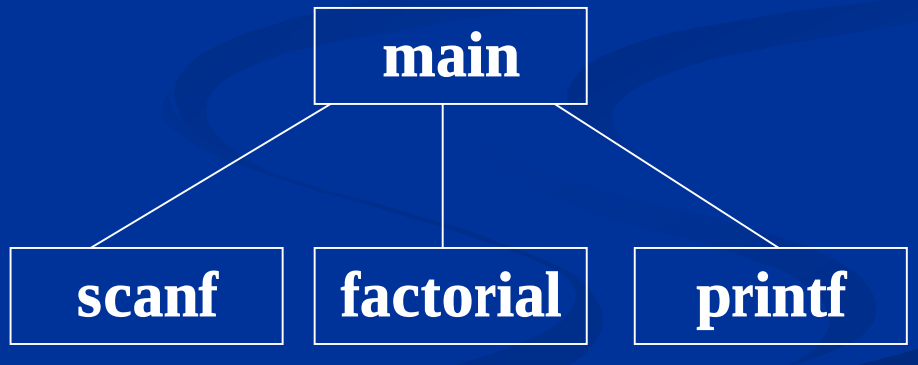
```
#include <stdio.h>
float  factorial ( int n )
{
    int  i;
    float  f = 1;
    for (i=1; i<=n; i++)
        f = f * i;
    return f;
}

int  main(void)
{
    int  k, m, n;
    float  fk, fm, fn;
    scanf("%d%d%d", &k, &m, &n);
    fk = factorial(k);
    fm = factorial(m);
    fn = factorial(n);
    printf("%f\n", fk/(fm+fn));
    return 0;
}
```

定义

调用

main 主调函数

factorial 被调函数



库函数　自定义函数　库函数

函数调用的方式

- 函数调用作为独立的语句
  - 例：printf("%f\n", fk/(fm+fn));
- 函数调用出现在表达式中
  - 例：fk = factorial(k);
- 函数调用作为另一个函数的参数
  - 例：printf("%f", factorial(k)/(factorial(m) + factorial(n)));

函数间的参数传递

- 定义函数时，根据参数的有无可分为：
  - 有参函数：即函数名后面括号中有参数，称为形式参数（形参）
  - 无参函数：即函数名后面括号中是空的（也可写**void**）
- 调用函数时，需按被调函数的定义形式使用：
  - 调用有参函数：需在函数名后面括号中指定参数，称为实际参数（实参）
  - 调用无参函数：函数名后面括号中为空（不能加**void**）
- 调用有参函数时，需进行参数间传递，传递规则如下：
  - 所有参数均以“传值调用”方式进行传递，且参数传递方式永远是从实参传递到形参

用函数解决问题示例

- 例：输入两个整数，计算平均值。要求用函数实现平均值的计算。
- 分析：定义函数时，需要明确哪些信息呢？
  - 函数是否需要外部给其传递数据？
  - 即是否需要参数，如需要，要几个？每个参数是什么数据类型？
  - 是否有返回值？如果有，是什么类型？
  - 起个合适的函数名

```
#include  <stdio.h>
double average(int x, int y)      //函数定义，x和y是两个形参
{
    double result;
    result = (x +y) / 2.0;
    return result;
}
int main(void )
{
    int  a, b;
    double ave;
    scanf("%d%d", &a, &b);
    ave = average(a, b);       //函数调用，a和b是两个实参
    printf("Average of %d and %d is %.1f\n", a, b, ave);
    return 0;
}
```

有参函数调用时要掌握的知识

- 当发生有参函数的调用时：
  - 形参才被分配存储空间，在调用结束后，形参所占的空间将被释放；
  - 实参需给形参传递数据，其值可以是常量，变量或表达式。不管是什么形式，都必须有确定的值；
  - 实参和形参的类型应相同或赋值相容；
  - 实参对形参的数据传递是“值传递”，即单向传递。
  - 实参和形参占用不同的内存单元，因此可同名。

例：交换两个数的值。

```
#include <stdio.h>
void swap(int a, int b)
{
    int  t;
    t=a;   a=b;   b=t;
    printf("交换后：  %d,%d\n", a, b);
}

int main(void)
{
    int  x=5,  y=9;
    printf("交换前：  %d  %d\n", x, y);
    swap(x, y);
    printf("函数调用结束后：  %d  %d\n", x, y);
    return 0;
}
```

形参

实参

执行过程：

x 5

y 9

交换前：5 9

交换后：9 5

如果在主函数中增加这一句，请问输出结果是什么？