



第5章 C函数

郑州大学软件学院/网络空间安全学院



Lecturer: 宋轩

Office : 行政楼-306

Email : songxuan@zzu.edu.cn

第5章 C函数——函数定义

问题的提出

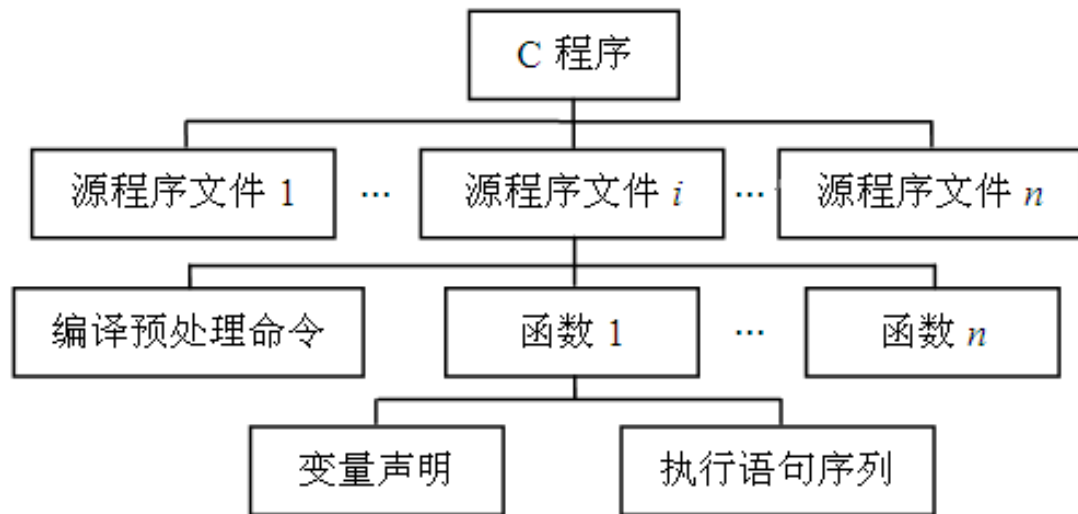
- 读多少行的程序能让你不头疼?
- 假如系统提供的函数printf()由10行代码替换, 那么你编过的程序会成什么样子?
 - 实际上一个printf()有上千行代码
- main()中能放多少行代码?
- 如果所有代码都在main()中, 怎么团队合作?
- 如果代码都在一个文件中, 怎么团队合作?

- 分而治之 (Divide and Conquer, Wirth, 1971)
 - 函数把较大的任务分解成若干个较小的任务，并提炼出公用任务

- 信息隐藏 (Information Hiding, Parnas, 1972)
 - 设计得当的函数可把具体操作细节对外界隐藏起来，从而使整个程序结构清楚
 - 使用函数时，不用知道函数内部是如何运作的，只按照我们的需要和它的参数形式调用它即可

- 函数是C语言中模块化编程的最小单位
 - 可以把每个函数看作一个模块（Module）
- 如把编程比做制造一台机器，函数就好比其零部件
 - 可将这些“零部件”单独设计、调试、测试好，用时拿出来装配，再总体调试。
 - 这些“零部件”可以是自己设计制造/别人设计制造/现成的标准产品。

- 若干相关的函数可以合并成一个“模块”
- 一个C程序由一个或多个源程序文件组成
- 一个源程序文件由一个或多个函数组成



- 函数生来都是平等的，互相独立的，没有高低贵贱和从属之分
 - `main()`稍微特殊一点点
 - C程序的执行从`main`函数开始
 - 调用其他函数后流程回到`main`函数
 - 在`main`函数中结束整个程序运行

■ 标准库函数

– ANSI/ISO C定义的标准库函数

- 符合标准的C语言编译器必须提供这些函数

- 函数的行为也要符合ANSI/ISO C的定义

– 第三方库函数

- 由其他厂商自行开发的C语言函数库

- 不在标准范围内，能扩充C语言的功能（图形、网络、数据库等）

■ 自定义函数

– 自己定义的函数

- 包装后，也可成为函数库，供别人使用

求一定范围内自然数的和

◆ 求出1到10、20到30和35到45的三个和

求和的代码

```
int i;  
int sum;
```

```
for (i = 1, sum = 0; i <= 10; i++){    sum += 1;  
}  
printf("%d到%d的和是%d\n", 1, 10, sum);
```

```
for (i = 20, sum = 0; i <= 30; i++){  
    sum += 1;  
}  
printf("%d到%d的和是%d\n", 1, 10, sum);
```

```
for (i = 35, sum = 0; i <= 45; i++){  
    sum += 1;  
}  
printf("%d到%d的和是%d\n", 1, 10, sum);
```

◆ 三段几乎一模一样的代码!

“代码复制” 是程序质量不良的表现

抽象为一个求和函数

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

```
//函数声明，可以省略变量名称
```

```
void sum(int begin,int end);
```

```
{
```

```
    int i;
```

```
    int sum=0;
```

```
    for (i=begin;i<=end;i++)
```

```
    {
```

```
        sum+=i;
```

```
    }
```

```
    printf("%d到%d的和是%d\n",begin,end,sum);
```

```
}
```

```
int main()
```

```
{
```

```
    sum(1,10);
```

```
    sum(20,30);
```

```
    sum(35,45);
```

```
}
```

如何进行改造输入两个任意的数，都能够进行计算

- 1.判定任意一个数是否是素数;
- 2.打印1-100以内的素数, 统计其个数, 并求和;

见示例程序ch05ppt06

什么是函数？

- 函数是一块代码，接收零个或多个参数，做一件事情，并返回零个或一个值
- 可以先想像成数学中的函数：
 - $y = f(x)$

什么是数学函数？

$$y = f(x)$$

一个变量
二个变量

.....

N个变量

自变量
与
因变量
的关系

一个变量
二个变量

.....

N个变量

程序设计中的函数

- 程序设计中的函数不局限于计算
 - 计算类，如打印阶乘表的程序.....
 - 判断推理类，如排序、查找.....

用函数解决问题的要点

- 分而治之
 - 函数把较大的任务分解成若干个较小的任务，并提炼出公用任务
- 复用
 - 程序员可以在其他函数的基础上构造程序，而不需要从头做起
- 信息隐藏
 - 设计得当的函数可以把具体操作细节对程序中不需要知道它们的那些部分隐藏掉，从而使整个程序结构清楚

C中的函数(Function)

- 说明:
 - 一个源程序文件由一个或多个函数组成。
 - 一个C程序由一个或多个源程序文件组成。
 - C程序的执行从main函数开始，调用其他函数后流程回到main函数，在main函数中结束整个程序运行。
 - 所有函数都是平行的，即函数定义时是互相独立的，一个函数并不从属于另一个函数。

函数定义 (definition)

- 返回值类型 函数名(类型 参数1, 类型 参数2,)

{

函数体

return 表达式; . . .

}

函数的返回值是通过
函数中的 return 语句
获得的。

- 如果没有参数, 则应该用void注明
- 如果不需要返回值, 则应该用void定义返回值类型
 - 返回值类型与return语句配合
- 当函数执行到return语句时, 就中止函数的执行, 返回到调用它的地方
- 函数内部可以定义只能自己使用的变量, 称内部变量。
 - 参数表里的变量也是内部变量

函数定义

返回类型

函数名

参数表

函数头

函数体

```
void sum(int begin, int end)
{
    int i;
    int sum = 0;
    for ( i=begin; i<=end; i++ ) {
        sum += i;
    }
    printf("%d到%d的和是%d\n", begin, end, sum);
}
```


函数参数

- 函数参数:
 - 形参(形式参数):
 - 在定义函数时, 定义函数名后面括号中的变量名
 - 实参(实际参数):
 - 在主调函数中调用一个函数, 调用函数名后面括号中的参数(或表达式)

调用函数

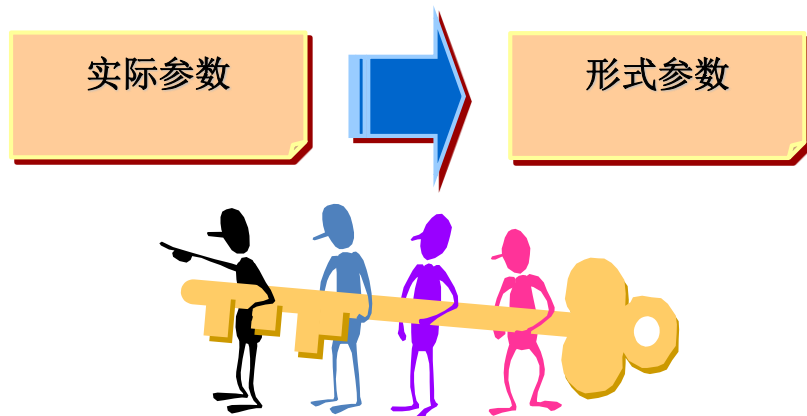
- 函数名(参数值);
- ()起到了表示函数调用用的重要作用
 - 即使没有参数也需要()
- 如果有参数, 则需要给出正确的数量和顺序
- 这些值会被按照顺序依次用来初始化函数 中的参数

```
sum(1,10);  
sum(20,30);  
sum(35,45);
```



```
void sum(int begin, int end)  
{  
    int i;  
    int sum = 0;  
    for ( i=begin; i<=end; i++ ) {  
        sum += i;  
    }  
    printf("%d到%d的和是%d\n", begin, end, sum);  
}
```

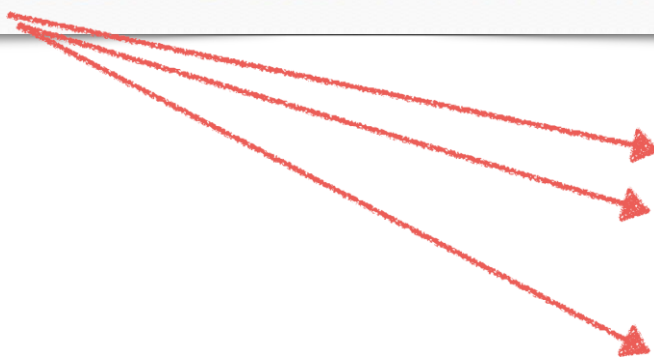
- 单向值传递
- 调用函数时，必须提供所有的参数
 - printf和scanf是采用变长变量表定义的函数，所以变量的个数不固定。
- 提供的参数个数、类型、顺序应与定义时相同



函数返回

- 函数知道每一次是哪里调用它，会返回到正确的地方。

```
void sum(int begin, int end)
{
    int i;
    int sum = 0;
    for ( i=begin; i<=end; i++ ) {
        sum += i;
    }
    printf("%d到%d的和是%d\n", begin, end, sum);
}
```



The diagram illustrates the flow of control when a function is called. Three red arrows originate from the closing curly brace '}' of the `sum` function definition. These arrows point to three separate lines of code: `sum(1, 10);`, `sum(20, 30);`, and `sum(35, 45);`. A fourth red arrow originates from the `return 0;` statement and points back to the opening curly brace '{' of the `sum` function definition, indicating the return path.

```
sum(1, 10);
sum(20, 30);
sum(35, 45);

return 0;
```

从函数中返回值

```
int max(int a, int b)
{
    int ret;
    if ( a>b ) {
        ret = a;
    } else {
        ret = b;
    }

    return ret;
}
```

- **return**停止函数的执行，并送回一个值。
- return;
- return 表达式;
- 一个函数里里可以出现多个return 语句

从函数中返回值

```
int max(int a, int b)
{
    int ret;
    if ( a>b ) {
        ret = a;
    } else {
        ret = b;
    }

    return ret;
}
```

```
int a,b,c;
a = 5;
b = 6;
c = max(10,12);
c = max(a,b);
c = max(c, 23);
c = max(max(c,a), 5);
printf("%d\n", max(a,b));
max(12,13);
```

- 可以赋值给变量
- 可以再传递给函数
- 甚至可以丢弃
- 有的时候要的是副作用

没有返回值的函数

- void 函数名(参数表)
- 不能使用带值的return
- 可以没有return
- 调用的时候不能做返回值的赋值

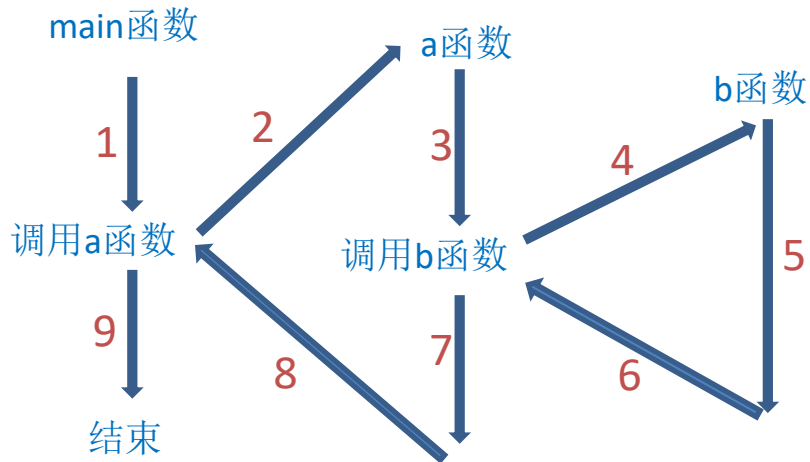
如果函数有返回值，则必须使用带值的return

```
void sum(int begin, int end)
{
    int i;
    int sum = 0;
    for ( i=begin; i<=end; i++ ) {
        sum += i;
    }
    printf("%d到%d的和是%d\n", begin, end, sum);
}
```

第5章 函数——函数调用

函数的嵌套调用

- ◆ C语言不允许嵌套定义函数，但是可以嵌套调用函数。
- ◆ 函数的嵌套调用是在调用一个函数的过程中，又调用另一个函数。



通过函数的嵌套调用，求两个正整数m，n的最小公倍数

- ◆ 假设计算m,n的最小公倍数和最大公约数的函数分别为sct(m,n)和gcd(m,n)
- ◆ 根据数学知识：两个正整数m，n的最小公倍数等于两数之积除以他们的最大公约数，记 $sct(m,n) = m * n / gcd(m,n)$
- ◆ main()中调用sct(m,n)，而sct(m,n)中调用gcd(m,n)

通过函数的嵌套调用，求两个正整数m，n的最小公倍数

```
int gcd(int m,int n)//求最大公约数
{
    int r=m;
    while (m%r !=0 || n%r !=0) r--;
    return r;
}
```

```
int sct(int m, int n)           //求最小公倍数
{
    int y;
    y=m*n/gcd(m,n);
    return y;
}
```

通过函数的嵌套调用，求两个正整数m，n的最小公倍数

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

```
int gcd(int m,int n)
```

```
{
```

```
.....
```

```
}
```

```
int sct(int m,int n)
```

```
{
```

```
.....
```

```
}
```

```
int main ()
```

```
{
```

```
int m,n,t;
```

```
scanf( "%d,%d" ,&m,&n);
```

```
t=sct(m,n) ;           //主函数调用sct()函数
```

```
printf( "%d和%d的最小公倍数%d/n" ,m,n,t);
```

```
return 0;
```

```
}
```

函数的递归调用

- 递归思想是指把一个规模较大的问题转化为形式相同但规模小一些的问题加以解决。
- 例如求e的n次幂。

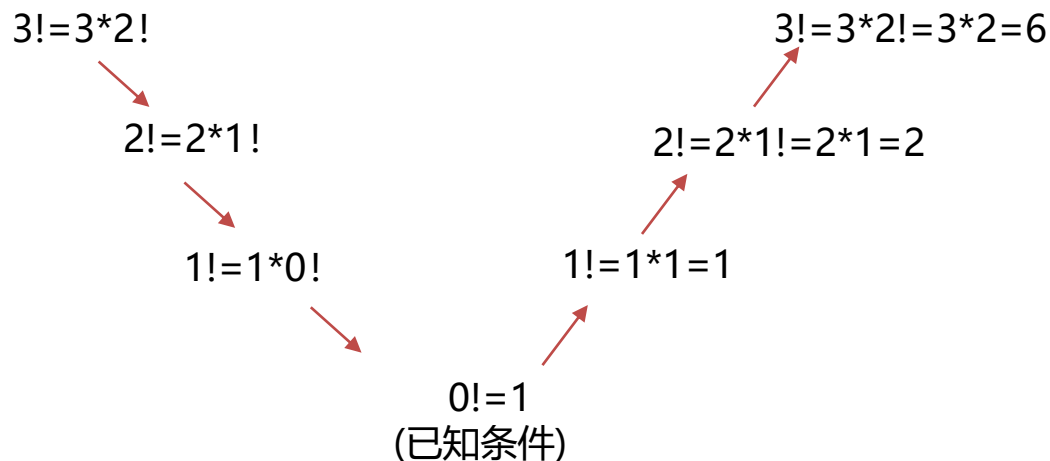
$$e^n = \begin{cases} 1 & \text{当 } n=0 \text{ 时} \\ e \times e^{n-1} & \text{当 } n=1, 2, 3 \dots \text{时} \end{cases}$$

函数的递归调用

- ◆ 在调用一个函数的过程中，又直接或间接地调用函数本身的情况，称为函数的递归调用。
- ◆ 两种形式的递归：
 - 直接递归： $f1$ 中调用 $f1$
 - 间接递归： $f1$ 中调用 $f2$ ， $f2$ 再调用 $f1$

示例：递归函数执行情况分析

例如，求 $3!$ 的递归算法的求值过程：



递归调用的过程可分为两个阶段：递归和回归

$$n! = \begin{cases} 1 & \text{当 } n=0 \text{ 时} \\ n \times (n-1)! & \text{当 } n=1, 2, 3 \dots \text{时} \end{cases}$$

求n!的递归函数定义如下:

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

可见，在递归函数中必须有:

- 1、递归结束条件
- 2、递归调用语句

示例：Fibonacci数列

Fibonacci数列的定义：

$f_1 = 1;$

$f_2 = 2;$

$f_n = f(n-1) + f(n-2) \quad n \geq 3$

示例：Fibonacci数列

```
//fibonacci数列函数
int fib(int n)
{
    if (n==1||n==2)
        return 1L;//递归出口
    else
        return fib(n-1)+fib(n-2); //递归调用
}
```

示例：Fibonacci数列

Fibonacci数列的定义：

$f_1 = 1;$

$f_2 = 2;$

$f_n = f_{n-1} + f_{n-2} \quad n \geq 3$

```
int fib(int n)
{
    if (n==1||n==2)
        return 1L;//递归出口
    else
        return fib(n-1)+fib(n-2); //递归调用
}
```

```
#include<stdio.h>
int fib(int );
int main()
{
    int i;
    for(i=1;i<=20;i++){
        printf("%d\t",fib(i));
        if (i%5==0) printf("\n");
    }
}
//最多能计算到n=? 46
```

第5章 C函数——变量的作用域

变量的生存空间和可见性

- **生存期：**变量占有存储空间的时间期限。
- **可见性：**是在程序中的哪个部分可以引用该变量。
- **变量的生存期是一个时间概念，而可见性是一个空间概念。**
- **变量可见一定存在，但存在不一定可见。**

为合理地定义和使用变量，应该了解变量属性

变量作用域

- 变量的作用域是指可以合法访问变量的范围，它是一个空间概念，由变量定义的位置来确定
- 变量定义的两种位置
 - 在所有函数之外
 - 在块内

变量作用域

◆ 在块内定义的变量称为**内部变量**，内部变量只在所在块中有效，故称**局部变量**。

- ⊙ 函数体是典型的块，所以在函数声明语句部分定义的变量是局部变量。
- ⊙ 形式参数只在函数中有效，也是局部变量。

```
int main ()
```

```
{
```

```
    int x,y;
```

```
    .....
```

```
    {
```

```
        int z;
```

```
        .....
```

```
        z=x-y
```

```
        .....
```

```
    }
```

```
}
```

```
int fun (int z)
```

```
{
```

```
    int x,y;
```

```
    .....
```

```
}
```

z在此范围内有效

x,y在此范围内有效

函数中的局部变量

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

```
void fun(int z);
```

```
int main()
```

```
{
```

```
    int x=1;
```

```
    fun(10);
```

```
    printf("x=%d\n",x);
```

```
    return 0;
```

```
}
```

x作用域

```
void fun(int z)
```

```
{
```

```
    int x,y;
```

```
    y=2*z;
```

```
    x=2;
```

```
    printf("x=%d,y=%d\n",x,y);
```

```
}
```

函数中的局部变量

```
#include<stdio.h>
void fun(int z);
int main()
{
    int x=1;
    fun(10);
    printf("x=%d\n",x);
    return 0;
}
```

x作用域

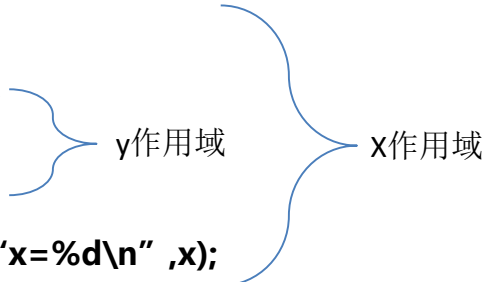
运行结果:
x=2,y=20
x=1

```
void fun(int z)
{
    int x,y;
    y=2*z;
    x=2;
    printf("x=%d,y=%d\n",x,y);
}
```

能改为printf("x=%d,y=%d\n" ,x,y);吗?

函数与复合语句中的局部变量

```
int main()
{
    int x;
    x=1;
    {
        int y;
        y=2;
        x=2;
    }
    printf( "x=%d\n" ,x);
    return 0;
}
```



运行结果:
x=2

- 程序运行进入这个块之前，其中的变量不存在，离开这个块，其中的变量就释放了

printf这句话改为printf("x=%d,y=%d\n",x,y); 会是什么结果，为什么？

函数与复合语句中的局部变量

- ◆ 块内定义了和块外同名的变量，则掩盖了块外的
- ◆ 不能在同一个块内定义同名的变量

运行结果：

x=2

x=1

```
int main()
{
    int x;
    x=1;
    {
        int x;
        x=2;
        printf( "x=%d\n" ,x);
    }
    printf( "x=%d\n" ,x);
    return 0
}
```

公共作用域范围外，外层同名变量
暂时被屏蔽

第5章 函数——变量的作用域

全局变量

- ◆ 在所有函数之外定义的变量称为**外部变量**。
- ◆ 外部变量的有效范围为从定义变量的位置开始到所在文件的结束，可以由其有效范围内的多个函数共用，因而也称为**全局变量**或**全程变量**。
- ◆ 默认值为**0**

```
#include <stdio.h>
int x=10,y=6;

int max(int a,int b)
{
    .....;
}

int main( )
{
    .....;
}
```

全局变量x和y的作用域


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

```
int x=10,y=6;
```

```
int max(int a,int b)
```

```
{
```

```
    int z;
```

```
    printf("%d\t%d\n",x,y);
```

```
    if(a>b) z=a;
```

```
    else z=b;
```

```
    return z;
```

```
}
```

```
int main()
```

```
{
```

```
    int x;
```

```
    x=3;
```

```
    printf("%d\n",max(x,y));
```

```
    printf("%d\t%d\t", x,y);
```

```
    return 0;
```

```
}
```

全局变量

x

10

y

6

max中的a

3

b

6

max中的z

6

全局变量x和y的作用域

main中的x

3

局部变量x的作用域

公共作用域范围内，全局变量
暂时被同名局部变量屏蔽

运行结果：

10 6

6

3

6

全局变量

- 全局变量在程序的整个执行期间都独自占有固定的内存单元，并保留其值，在整个程序的运行期（不管在函数内外），总是存在。
- 可以被其作用域范围内的多个函数（若有的话）所使用和修改。

```
#include <stdio.h>
int x,y;
void swap()
{
    int t;
    t=x;
    x=y;
    y=t;
}

int main()
{
    scanf("%d%d",&x,&y);
    swap();
    printf("x=%d,y=%d\n",x,y);
    return 0;
}
```

利用全局变量交换变量值

若从键盘输入x,y的值为
3 8
则输出结果为：
x=8,y=3

选择变量的原则

- 当变量只在某个函数或复合语句内使用时，不要定义成全局变量。
- 当多个函数引用同一个变量时，在这些函数上面定义全局变量，而且定义部分尽量靠近这些函数。

第5章 函数——变量的存储类型

变量的存储类别

- 存储类别指变量在内存中存储的方式，分为
 - ◆ 静态存储方式 只在程序编译的时候分配存储空间，在程序运行期间一直占有固定的存储单元，直到程序运行的结果
 - ◆ 动态存储方式 是在程序运行期间，根据需要进行动态的分配和释放存储空间的方式
- 全局变量、加static修饰的局部变量属于静态存储方式
- 形式参数、未知static修饰的局部变量属于动态存储方式

指定存储类型的变量定义格式：

存储类型说明符 数据类型名 变量名表；

四个存储类型说明符：告诉编译程序如何保存有关变量

- auto (自动)
- extern (外部)
- register (寄存器)
- static (静态)

auto **int** c1,c2; //说明c1,c2为自动变量，存放在内存的动态存储区中

register int i,j; //说明i,j为寄存器变量，存放在寄存器中

static float x,y,z; //说明x,y,z为静态变量，存放在内存的静态存储区中

auto变量（自动变量）

在函数（或复合语句）内部定义，格式如下：

[auto] 数据类型名 变量名表;

例：auto int a,b ;

由于auto可省略，所以，函数内所有未加存储类别说明符定义的局部变量均为自动变量。

auto变量（自动变量）

- 自动变量是局部变量；函数的形参是一种自动变量。
- 自动变量是C程序中使用最多的变量。好处是：用之则建，用完即撤，节省存储空间。

```
#include <stdio.h>
void fun(int y)
{
    auto int x=1; //每次调用时，都要重新初始化；
    x++;
    printf( "%d:x=%d\n",y,x);
}
```

```
int main()
{
    int i; // auto型
    for(i=0;i<3;i++)
        fun(i);
    return 0;
}
```


register变量（寄存器变量）

在函数（或复合语句）内部定义，格式如下：

```
register 数据类型名 变量名表;
```

- 变量存储在CPU的寄存器，操作速度远快于存储在内存中的普通变量。
- register只能用于局部变量和函数形参。
 - 对于循环次数较多地循环控制变量，循环体内反复使用的变量和形参均可定义为寄存器变量。
- 现代编译器都有优化功能，自动将频繁使用的变量放在寄存器中，不需要编程者指定，因此，不需要使用register关键字

static局部变量（静态局部变量）

在函数（或复合语句）内部定义，格式如下：

static 数据类型名 变量名表：

例：static int a,b;

与自动变量相同的：静态局部变量的作用域也是从其他定义的位置其，到函数体结束（或复合语句）结束为止。

static局部变量（静态局部变量）

例：观察静态局部变量和动态局部变量在调用过程中的情况

```
#include <stdio.h>
void fun()
{
    int x=1;//省略auto, 每次调用时, 都要重新初始化
    x++;
    printf("x=%d\n",x);
}
int main()
{
    int i;
    for(i=0;i<3;i++)
        fun();
    return 0;
}
```

运行结果：

x=2

x=2

x=2

static局部变量（静态局部变量）

与自动变量不同的是（运行步骤）：

- 静态局部变量的初值是在编译时赋予的（仅赋值一次），及在程序运行是它已有初值。
- 定义该变量的函数在每次调用时，使用上次函数调用结束时留下来的值。

```
#include <stdio.h>
void fun()
{
    static int x=1;
    x++;
    printf( "x=%d\n" ,x);
}
int main()
{
    int i;
    for(i=0;i<3;i++)
        fun();
    return 0;
}
```

fun中的x

4

运行结果：

X=2

X=3

X=4

static局部变量（静态局部变量）

与自动变量不同的是（运行步骤）：

- 如果基本类型的静态局部变量在定义时未赋初值，**编译时会自动初始化为0。**
- 在整个程序运行期间，静态局部变量在内存的静态存储区中占据了固定的存储单元，即在定义该变量的函数被调用结束后其所在占据的存储单元并不释放，直至整个程序运行结束。

```
//例题：统计进入fun函数的次数
#include<stdio.h>
void fun()
{
    static int x;
    x++;
    printf( "第%d次进入函数\n",x);
}
```

全局变量的存储类型

知识回顾：全局变量是在所有函数的外部定义的变量，作用域为从变量定义的位置开始，到本源文件结束为止。

指定存储类型的格式：

存储类型说明符 数据类型名 变量名表

- **extern (外部)**：扩展变量的作用域，使之可被程序中各个函数所引用。
- **static (静态)**：表示该变量只可以被定义它的文件中的各函数所引用。

全局变量的存储类型

- 缺省类别是extern,故又称为“外部变量”，静态存储方式，编译时将全局变量分配在静态存储区
- 初值是在编译时赋予的（仅赋予一次），若在定义时未赋予初值，编译时会自动被初始化为0（字符型为‘0’）

选择变量的原则

```
#include<stdio.h>
extern int x,y;    //也可以写成: extern x,y;
int main()
{
    int max(int a,int b);
    printf("%d",max(x,y));
    return 0;
}
int x=111,y=10;
int max(int a,int b)
{
    int c;
    if(a>b) c=a;else c=b;
    return c;
}
```

扩散作用域:

如果全局变量不在文件开头定义，其有效范围只限于定义处到文件尾。

如果在定义点之前的函数想引用该全局变量，则应该在引用之前用关键字 **extern** 对该变量引用性声明，以告诉编译器该变量在本文件的某处已经被定义。

选择变量的原则

- 当变量只在某个函数或复合语句内使用时，不要定义成全局变量。
- 当多个函数引用同一个变量时，在这些函数上面定义全局变量，而且定义部分尽量靠近这些函数。

第5章 函数——小结

本章小结

- 在C语言中可以从不同角度对函数分类：
 - ◆ 库函数：由C系统提供的函数;
 - ◆ 用户定义函数：由用户自己定义的函数;
 - ◆ 有返回值的函数：向主调函数返回函数值，应说明函数类型（即返回值的类型);
 - ◆ 无返回值的函数：不返回函数值，函数类型为空类型void;
 - ◆ 有参函数：主调函数向被调函数传递数据;
 - ◆ 无参函数：主调函数没有向被调函数传递数据;

本章小结

◆ 函数定义的一般形式

函数类型 函数名 ([形参表])

{

声明与定义部分

语句部分

}

◆ 函数声明的一般形式

函数类型 函数名 ([形参表]) ;

◆ 函数调用的一般形式：函数名 ([实参表])

◆ 在C语言中，不允许嵌套定义函数，但是允许函数的嵌套调用或递归调用。

本章小结

- 函数的参数分为形参和实参两种，形参出现在函数定义中，实参出现在函数调用中，发生函数调用时，把实参的值传递给形参，**形参**是变量，必须给以类型说明；**实参**可以是常量，变量，表达式，函数等，无论是何种类型的量，必须具有确定的量。
- 函数的返回值由return语句实现，其格式为：
return [表达式]；
- 函数先计算return表达式的值并转换为函数类型，然后返回到主调函数中替换函数调用。

本章小结

可从三个方面对变量分类

- 变量的**数据类型**是其操作性，决定变量可参与的运算以及占内存的大小，在前面已经介绍。
- 变量的**作用域**是指变量在程序中的有效范围，即从空间角度将变量分为全局变量和局部变量。
- 变量的**存储类别**是指变量在内存中的存储方式，分为静态存储和动态存储，决定了变量的生存期。

Questions & Answers

常用函数库的头文件

头文件	解释
<assert.h>	包含了为增加辅助程序调试的诊断功能所需的相关信息
<ctype.h>	包含了测试具有某种属性字符的函数、能够将小写字母转换成大写字母或者反过来转换的函数的函数原型
<errno.h>	定义用于报告错误条件的宏
<float.h>	包含了系统中浮点数大小的上下限
<limits.h>	包含了系统中整数大小的上下限
<locale.h>	包含了能够根据正在运行的当前现场来修改程序的函数原型和其他信息。程序现场的标记使得计算机系统能够处理在表示诸如日期、时间、现金总数及涉及整个世界的巨大数值这样的数据时不同的习惯用法
<math.h>	包含了数学函数库的函数原型
<setjmp.h>	包含了能够绕过通常的程序调用和返回序列的函数的函数原型
<signal.h>	包含了处理在程序运行中可能产生的各种条件的函数原型和宏
<stdarg.h>	定义处理传递给一个形参个数和类型都是未知的函数的一组实参的宏
<stddef.h>	包含了C语言用来执行计算的公共类型的定义
<stdio.h>	包含了标准输入/输出库函数的函数原型，及其所需的相关信息
<stdlib.h>	包含了将数值转换成文本或者文本转换成数值函数、内存分配函数、随机数函数和其他工具函数的函数原型
<string.h>	包含了字符串处理函数的函数原型
<time.h>	包含了用于处理时间和日期的函数原型和数据类型

常用数学库函数

函 数	说 明	例 子
sqrt(x)	x 的平方根	sqrt(900.0) 的值是 30.0 sqrt(9.0) 的值是 3.0
cbrt(x)	x 的立方根 (仅在 C99 和 C11 中提供)	sqrt(27.0) 的值是 3.0 sqrt(-8.0) 的值是 -2.0
exp(x)	指数函数 e^x	exp(1.0) 的值是 2.718282 exp(2.0) 的值是 7.389056
log(x)	x 的自然对数 (以 e 为底)	log(2.718282) 的值是 1.0 log(7.389056) 的值是 2.0
log10(x)	x 的对数 (以 10 为底)	log10(1.0) 的值是 0.0 log10(10.0) 的值是 1.0 log10(100.0) 的值是 2.0
fabs(x)	以浮点数表示的 x 的绝对值	fabs(13.5) 的值是 13.5 fabs(0.0) 的值是 0.0 fabs(-13.5) 的值是 13.5
ceil(x)	对 x 向上取整得到不小于 x 的最小整数	ceil(9.2) 的值是 10.0 ceil(-9.8) 的值是 -9.0
floor(x)	对 x 向下取整得到不大于 x 的最大整数	floor(9.2) 的值是 9.0 floor(-9.8) 的值是 -10.0
pow(x, y)	x 的 y 次幂 (x^y)	pow(2, 7) 的值是 128.0 pow(9, .5) 的值是 3.0
fmod(x, y)	以浮点数表示的 x/y 的余数	fmod(13.657, 2.333) 的值是 1.992
sin(x)	x 的正弦值 (x 以弧度表示)	sin(0.0) 的值是 0.0
cos(x)	x 的余弦值 (x 以弧度表示)	cos(0.0) 的值是 1.0
tan(x)	x 的正切值 (x 以弧度表示)	tan(0.0) 的值是 0.0