第六章 函数

郎大鹏

第六章 函数

第六章 函数

6.1 高效程序的编写方法

程序员在设计一个复杂的应用程序时，往往也是把整个程序划分为若干功能较为单一的程序模块，然后分别予以实现，最后再把所有的程序模块像搭积木一样装配起来，这种在程序设计中分而治之的策略，被称为模块化程序设计方法。

函数是程序的基本组成单位，因此可以很方便地用函数作为程序模块来实现C语言程序。利用函数，不仅可以实现程序的模块化，程序设计得简单和直观，提高了程序的易读性和可维护性，

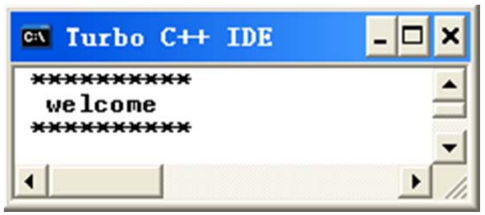
6.1 高效程序的编写方法

由于采用了函数模块式的结构，C语言易于实现结构化程序设计。使程序的层次结构清晰，便于程序的编写、阅读、调试。

例6.1 输出两行“*”和一行信息，如下所示：

```
**********
 welcome
**********
```

```
#include<stdio.h>
void main()
{
void star();                /*对star函数进行声明*/
 void message();            /*对message函数进行声明*/
 star();                    /*调用star函数*/
 message();                /*调用message函数*/
 star();                    /*调用star函数*/
}
void star()                  /*定义star函数*/
{
 printf(" **********\n");
}
void message()             /*定义message函数*/
{
 printf("  welcome\n");
}
```



6.1 高效程序的编写方法

1．一个源程序文件由一个或多个函数组成。一个源程序文件以源文件为单位进行编译，而不是以函数为单位进行编译。

2．一个C程序由一个或多个源程序文件组成。对较大的程序，一般不希望全放在一个文件中，而将函数和其它内容（如预定义）分别放到若干个源文件中，再由若干源文件组成一个C程序。

3．C程序的执行从main函数开始，调用其它函数后流程回到main函数，在main函数中结束整个程序的运行。main函数是系统定义的。

4．所有函数都是平行的，一个函数并不从属于另一函数，即函数不能嵌套定义（这是和PASCAL不同的），但可以互相调用，但不能调用main函数。

6.1 高效程序的编写方法

5．从用户使用的角度看，函数有两种：

（1）标准函数，即库函数。这是由系统提供的，用户不必自己定义这些函数，可以直接使用它们。应该说明，每个系统提供的库函数的数量和功能不同，当然有一些基本的函数是共同的。

（2）用户自己定义的函数，以解决用户的专门需要。

6. 从函数的形式看，函数分两类

（1）无参函数。如例6．1中的star和message就是无参函数。在调用无参函数时，主调函数并不将数据传送给被调用函数，一般用来执行指定的一 组操作（例如例6．1那样，以star函数的作用就是输出18个星号）。无参函数可以带回或不带回函数值，但一般以不带回函数值的居多。

(2)有参函数。在调用函数时，在主调函数和被调用函数之间有参数传递，也就是说，主调函数可以将数据传给被调用函数使用，被调用函数中的数据也可以带回来供主调函数使用。

第六章 函数

6.2 函数的定义

6.2.2　无参函数的定义

类型标识符　函数名（）
　　{说明部分
　　语句　}

```
void Hello()
   {
      printf ("Hello,world \n");
}
```

函数的类型实际上是函数返回值的类型。
函数名是由用户定义的标识符。
函数名后有一个空括号，其中无参数，但括号不可少。
{ }中的内容称为函数体。在函数体中声明部分，是对函数体内部所用到的变量的类型说明。
　在很多情况下都不要求无参函数有返回值，此时函数类型符可以写为void。

6.2 函数的定义

6.2.2 有参函数的定义

类型标识符 函数名（形式参数表列）
{说明部分
语句 }

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

第一行说明max函数是一个整型函数，其返回的函数值是一个整数。形参为a,b,均为整型量。a,b的具体值是由主调函数在调用时传送过来的。在{ }中的函数体内，除形参外没有使用其它变量，因此只有语句而没有声明部分。在max函数体中的return语句是把a(或b)的值作为函数的值返回给主调函数。有返回值的函数中至少应有一个return语句。

6.2 函数的定义

6.2.2 有参函数的定义

在C程序中，一个函数的定义可以放在任意位置，既可放在主函数main之前，也可放在main之后。

```
include <stdio.h>
void main()
{
    int max(int a,int b);
    int x,y,z;
    printf("input two numbers:\n");
    scanf("%d%d",&x,&y);
    z=max(x,y);
    printf("maxmum=%d",z);
}
int max(int a,int b)
{
    if(a>b)return a;
    else return b;
}
```

6.2 函数的定义

空函数

什么工作也不做，没有任何实际作用。在主调函数中写上fun1()表明“这里要调用一个函数”，而现在这个函数没有起作用，等以后扩充函数功能时补充上。

C语言可以有“空函数”，它的形式为：

```
类型说明符  函数名（）
{      }
```

例如

```
fun1( )
{ }
```

第六章 函数

6.3 函数间数据的传递方法

6.3.1 形式参数和实际参数

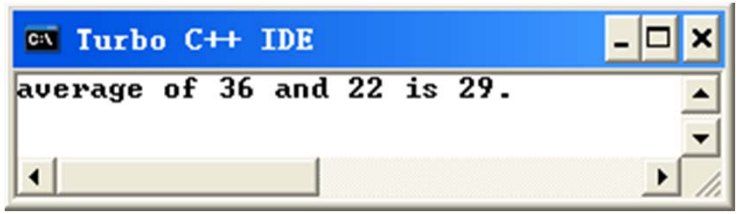
形参和实参的功能是作数据传送。发生函数调用时，主调函数把实参的值传送给被调函数的形参从而实现主调函数向被调函数的数据传送。

1、形参变量只有在被调用时才分配内存单元，在调用结束时，即刻释放所分配的内存单元。
2、实参在使用时，应预先用赋值，输入等办法使实参获得确定值。
3、实参和形参在数量上，类型上，顺序上应严格一致，否则会发生类型不匹配的错误。
4、函数调用中发生的数据传送是单向的。即只能把实参的值传送给形参，而不能把形参的值反向地传送给实参。

6.3 函数间数据的传递方法

例6.3 计算两个整数的平均数的函数

```
#include<stdio.h>
int average(int x,int y) {
/*函数定义*/
 int z ;
 z=(x+y)/2 ;
 return(z) ;
 }
void main(){
 int a=36;
 int b=22;
 int c=average(a,b);
          /*函数调用*/
 printf("average of %d and %d is
%d.\n",a,b,c);
}
```



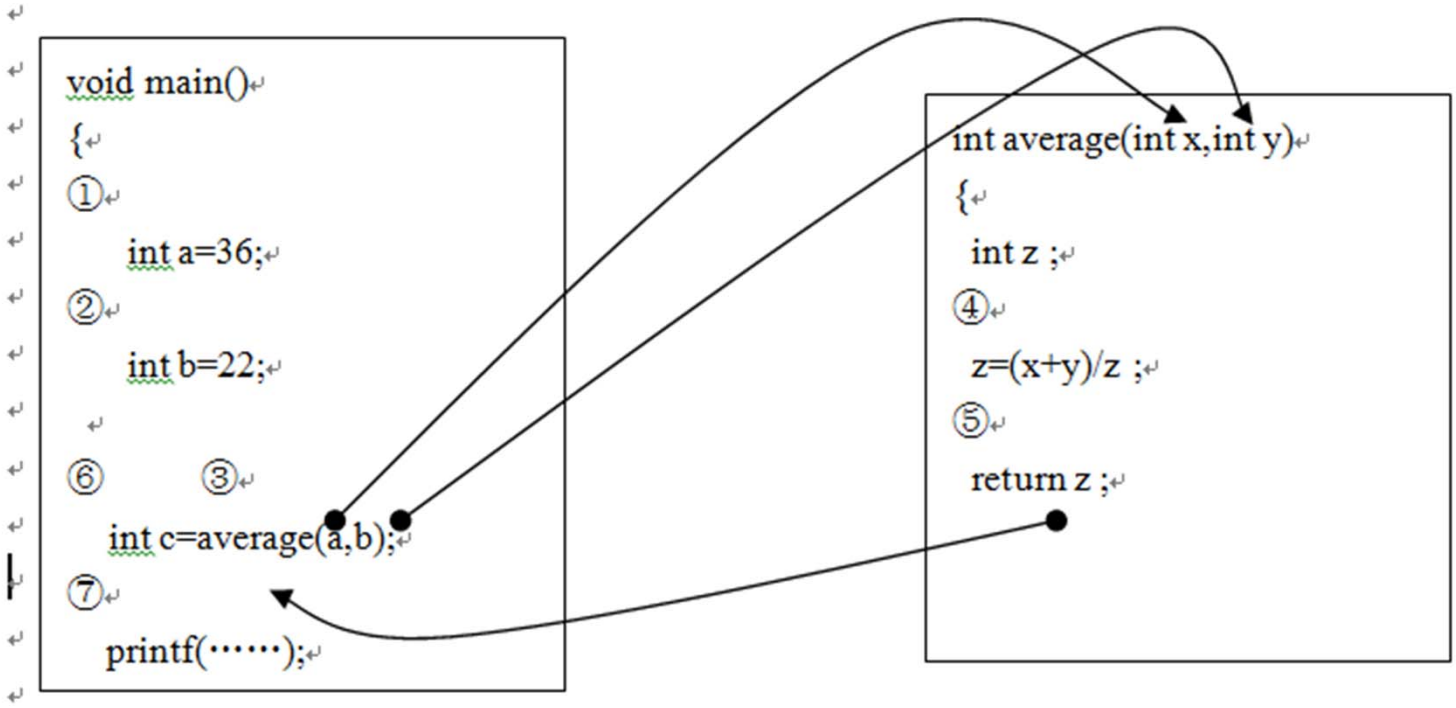
6.3 函数间数据的传递方法

6.3.1 形式参数和实际参数

说明：
average函数通过参数传递知道了要计算的是36和22的平均值。在进行函数调用时，把变量a和b的值作为参数提供给average函数。这时程序开始执行average函数，并把a和b的值分别传递给average函数中定义的参数x与y，这个过程就是参数传递。函数内接收数据的参数叫形式参数，简称形参，调用函数提供的参数叫实际参数，简称实参。本例中的x和y就是形参，a和b是实参。

6.3 函数间数据的传递方法

上述调用过程可以用下图说明。



6.3 函数间数据的传递方法

6.3.2 函数的返回值

通常，希望通过函数调用使主调函数能得到一个确定的值，这就是函数的值，也就是函数的返回值。

1．函数的返回值是通过函数中的return语句获得的。 return语句将被调用函数中的一个确定值带回主调函数中去。如果不需要从被调用函数带回函数值可以不要return语句。

return后面的值可以是一个表达式。例如，例6．2中的函数max可以改写如下：

```
int max(int a,int b)
{
  return（x>y？x： y）；
}
```

6.3 函数间数据的传递方法

6.3.2 函数的返回值

2．函数值的类型。既然函数有返回值，这个值当然应属于某一个确定的类型，应当在定义函数时指定函数值的类型。例如：

int max（int x，int y） 函数值为整型

char letter（char c1，char c2）函数值为字符型

double min（double x，double y） 函数值为双精度型

3．如果函数值的类型和return语句中表达式的值不一致，则以函数值的类型为准。对数值型数据，可以自动进行类型转换。即函数类型决定返回值的类型。

6.3 函数间数据的传递方法

例6.4 将例6.2稍作改动（注意是变量的类型改动）。

```
#include <stdio.h>
void main（）
 {
float  a，b；
int c;
scanf(“%f,%f”,&a,&b);
c＝max（a,b）；
printf（”Max is %d”,c）；
}
int max（float x, float y）;
 {
float  z ;
z = x ＞ y ？ x ： y ；
return（z）；
}
```

运行情况如下：

1.5，2.5

Max is 2

6.3 函数间数据的传递方法

4．如果被调用函数中没有return语句，并不带回一个确定的、用户所希望得到的函数值，但实际上，函数并不是不带回值，而只是不带回有用的值，带回的是一个不确定的值。

```
 {
int a，b,c;
        a=star()
;       b＝message（）；
        c＝star（）；
        printf(:%d,%d,%d”,a,b,c);
}
```

运行时除得到和例6．1一样的结果外，还可以输出a,b, c值。当然，a,b,c 的值不一定有实际意义.

6.3 函数间数据的传递方法

5. 为了明确表示“不带回值”，可以用“void”定义“无类型”（或称“空类型”），例如，例6.1中的定义为：

```
void star（）
{ … }
void message（）
{ … }
```

这样，系统就保证不使函数带回任何值，即禁止在调用函数中使用被调用函数的返回值。如果star和messag函数定义为void型，则下面的用法就是错误的：

```
a=star( ) ;
b=message( );
```

第六章 函数

6.4 函数的调用

6.4.1 函数调用的一般形式

C语言中，函数调用的一般形式为：

函数名（实参表列）；

如果是调用无参函数则实参表列可以没有，但括弧不能省略

在C语言中，可以用以下几种方式调用函数：

1、函数表达式：函数作为表达式中的一项出现在表达式中，以函数返回值参与表达式的运算。这种方式要求函数是有返回值的。

2、函数语句：函数调用的一般形式加上分号即构成函数语句。

3、函数实参：函数作为另一个函数调用的实际参数出现。这种情况是把该函数的返回值作为实参进行传送，因此要求该函数必须是有返回值的。

6.4 函数的调用

在函数调用中还应该注意的一个问题是求值顺序的问题。所谓求值顺序是指对实参表中各量是自左至右使用呢，还是自右至左使用。

例6.5

```
        #include <stdio.h>
int  f（int a，int b）;
        {
int  c;
            if（a>b）c=1;
            else  if（a==b）c=0;
            else  c =-1;
            return（c）;
}
void main（ ）
          {
int  i=2，p;
            p=f(i,++i)
            printf(“%d”,p);
                                  }
```

6.4 函数的调用

如果本意是按自左 而右顺序求实参的值的，

可以改写为：

```
j=i;
k=++i;
p=f(j,k);
.....
```

如果本意是自右而左求实参的值的，可改写为：

```
j=++i;
p=f(i,j)
```

这种情况在printf函数中也同样存在，如

```
printf（”%d,%d”,i,i++）;
```

6.4 函数的调用

6.4.1　函数的声明和函数原型

在主调函数中调用某函数之前应对该被调函数进行说明（声明），函数声明的形式为：

类型说明符　函数名(类型 形参，类型 形参…);

或为：

类型说明符　函数名(类型，类型…);

括号内给出了形参的类型和形参名，或只给出形参类型。这便于编译系统进行检错，以防止可能出现的错误。

6.4 函数的调用

6.4.1 函数的声明和函数原型

例6.2 main函数中对max函数的说明为：

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

或写为:

```
int max(int,int);
```

1、如果被调函数的返回值是整型或字符型时，可以不对被调函数作说明，而直接调用。这时系统将自动对被调函数返回值按整型处理。

2、当被调函数的函数定义出现在主调函数之前时，在主调函数中也可以不对被调函数再作说明而直接调用。

6.4 函数的调用

如在所有函数定义之前，在函数外预先说明了各个函数的类型，则在以后的各主调函数中，可不再对被调函数作说明。

例如：

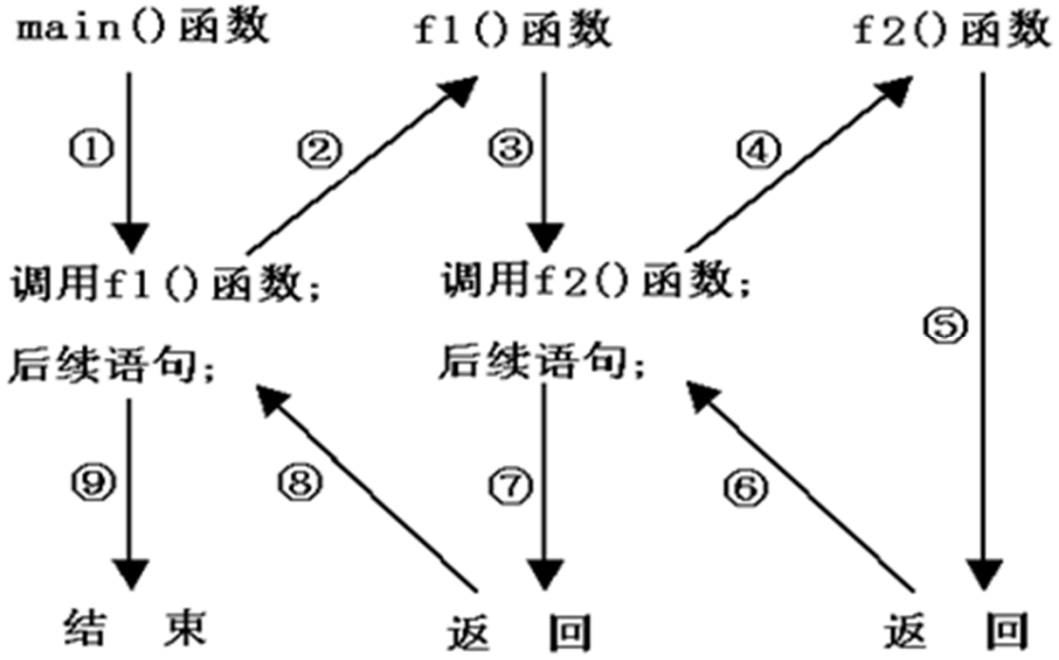
```
char str(int a);
float f(float b);
main()
    {
     ……
    }
char str(int a)
    {
     ……
    }
float f(float b)
    {
     ……
    }
```

其中第一，二行对str函数和f函数预先作了说明。因此在以后各函数中无须对str和f函数再作说明就可直接调用。

第六章 函数

6.5 函数的嵌套调用

C语言的函数定义都是互相平行、独立的，也就是说在定义函数时，一个函数内不能包含另一个函数。即C语言不能嵌套定义函数，但可以嵌套调用函数，也就是说，在调用一个函数的过程中又调用另一个函数。



6.5 函数的嵌套调用

例6.6 计算（1！）^{2}+(2!)^2+（3！）^{2}

```
#include<stdio.h>
long  f1(int p) {    /*计算1到3阶乘的2次方*/
    long l;
        long f2(int);    /*函数声明*/
        l=f2(p);         /*对函数f2的调用*/
        return l*l;
}
long f2(int q) {        /*计算1到3的阶乘*/
    long c=1;
       int i;
       for(i=1;i<=q;i++)
       c=c*i;
       return(c);
}
void main(){
    int i;
    long s=0;
    for (i=1;i<=3;i++)
    s=s+f1(i);          /*对函数f1的调用*/
    printf("\ns=%ld\n",s);
 }
```

6.5 函数的嵌套调用

例6.7　计算s=1^k+2^k+3^k+.....+N ^{k}

```
/*功能：函数的嵌套调用*/
#define K 4
#define N 5
long  f1(int n,int k) {   /*计算n的k次方*/
   long power=n;
   int i;
   for(i=1;i<k;i++)  power *= n;
   return power;
 }
long  f2(int n,int k){ /*计算1到n的k次方之累加和*/
    long sum=0;
    int i;
    for(i=1;i<=n;i++)  sum += f1(i, k);
         /*以i、k作实参，循环调用函数f1*/
     return sum;
  }
```

```
void main()
  {
    printf("Sum of %d powers of integers from 1 to %d = ",K,N);
    printf("%d\n",f2(N,K));
       /*以N、K作实参，调用函数f2*/
  }
```

第六章 函数

形参&&实参

函数的形参和实参具有以下特点：

•1. 形参变量只有在被调用时才分配内存单元，在调用结束时，即刻释放所分配的内存单元。因此，形参只有在函数内部有效。函数调用结束返回主调函数后则不能再使用该形参变量。

•2. 实参可以是常量、变量、表达式、函数等，无论实参是何种类型的量，在进行函数调用时，它们都必须具有确定的值，以便把这些值传送给形参。因此应预先用赋值，输入等办法使实参获得确定值。

•3. 实参和形参在数量上，类型上，顺序上应严格一致，否则会发生类型不匹配”的错误。

•4. 函数调用中发生的数据传送是单向的。即只能把实参的值传送给形参，而不能把形参的值反向地传送给实参。 因此在函数调用过程中，形参的值发生改变，而实参中的值不会变化。

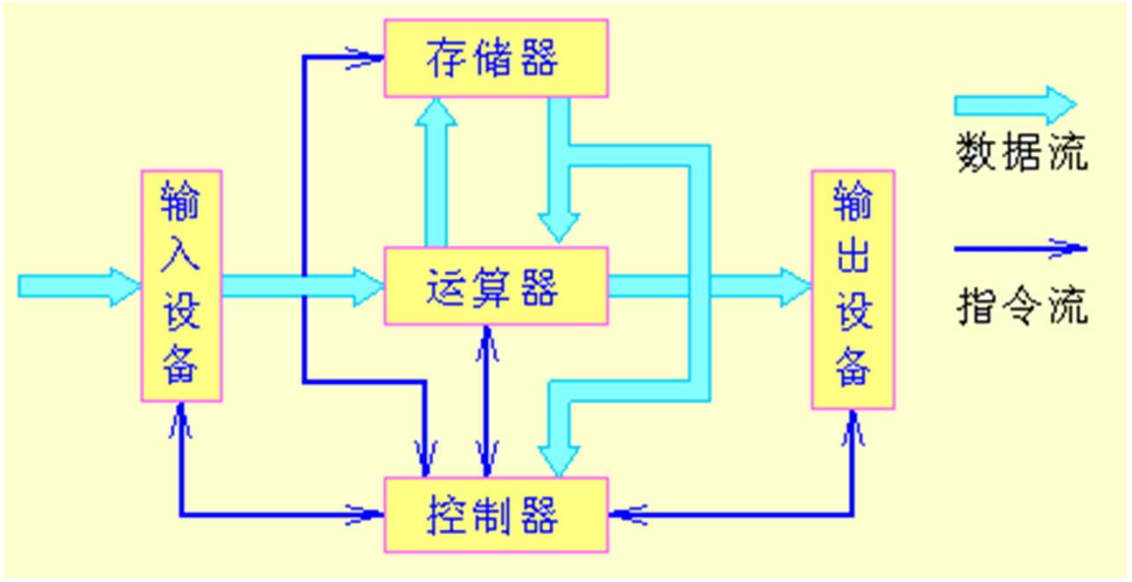
printf(“例子\n”);

```
#include <stdio.h>
main()
{
    int n;
    printf("input number\n");
    scanf("%d",&n);
    s(n);
    printf("n=%d\n",n);
}
```

```
int s(int n)
{
    int i;
    for(i=n-1;i>=1;i--)
    n=n+i;
    printf("n=%d\n",n);
}
```

计算机的基础知识

约翰·冯·诺依曼
计算机之父

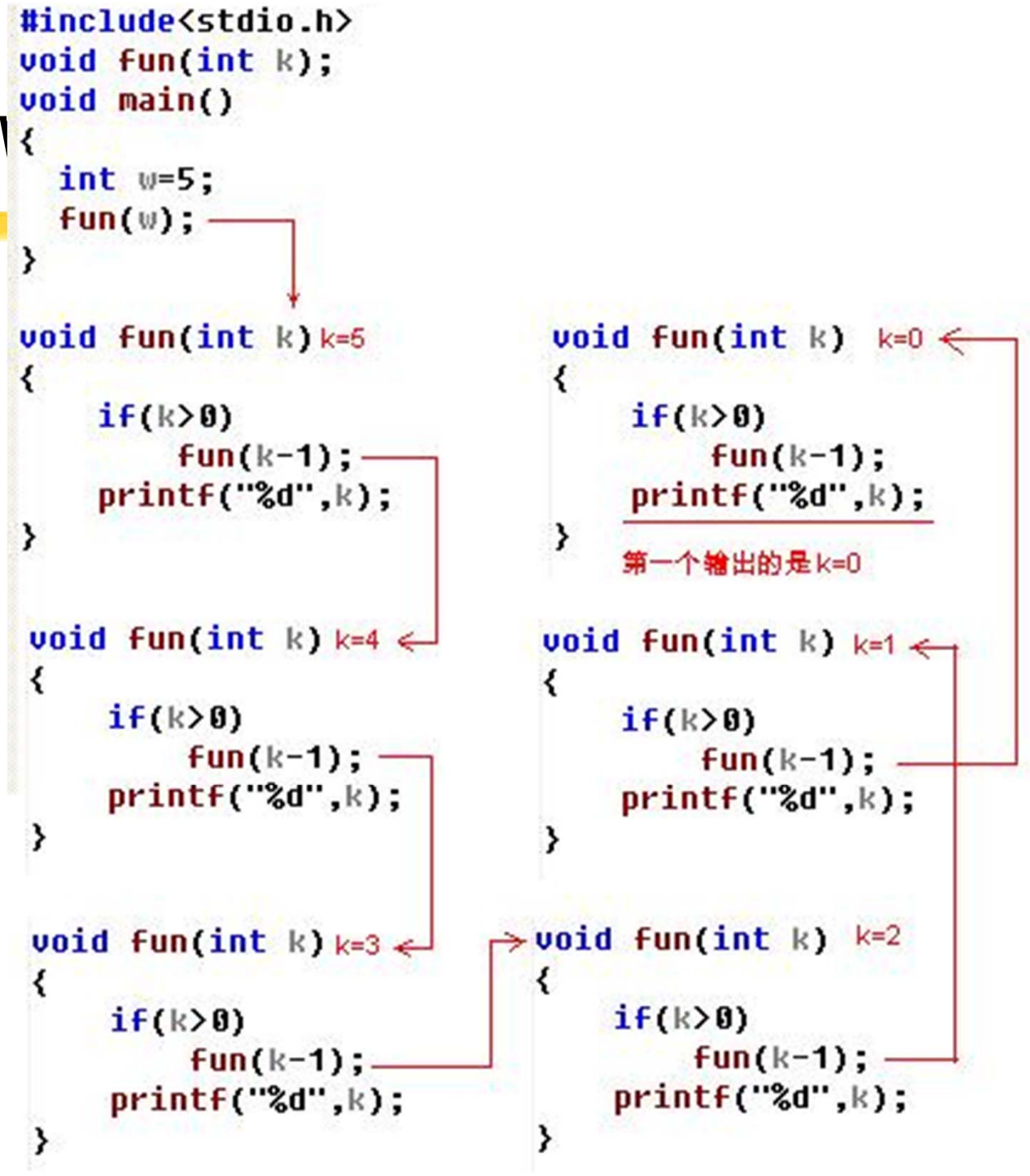


printf(“例子\n”);

```
#include<stdio.h>
void fun(int k);
void main()
{
  int w=5;
  fun(w);
}
```

```
void fun(int k)
{
 if(k>0)
  fun(k-1);
 printf("%d",k);
}
```

printf("例子\


#include<stdio.h>
void fun(int k);
void main()
{
int w=5;
fun(w);
}
void fun(int k) k=5
{
if(k>0)
fun(k-1);
printf("%d",k);
}
void fun(int k) k=4
{
if(k>0)
fun(k-1);
printf("%d",k);
}
void fun(int k) k=3
{
if(k>0)
fun(k-1);
printf("%d",k);
}
void fun(int k) k=2
{
if(k>0)
fun(k-1);
printf("%d",k);
}
void fun(int k) k=1
{
if(k>0)
fun(k-1);
printf("%d",k);
}
void fun(int k) k=0
{
if(k>0)
fun(k-1);
printf("%d",k);
}
第一个输出的是k=0

6.6 函数的递归调用

函数的递归调用是指，一个函数在它的函数体内，直接或间接地调用它自身。这种函数称为递归函数。C 语言允许函数的递归调用。在递归调用中，调用函数又是被调用函数，执行递归函数将反复调用其自身。每调用一次就进入新的一层。

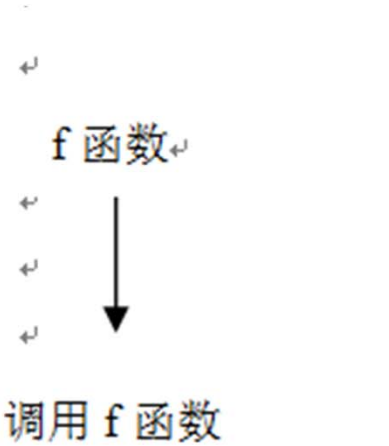


图 6.7　直接调用

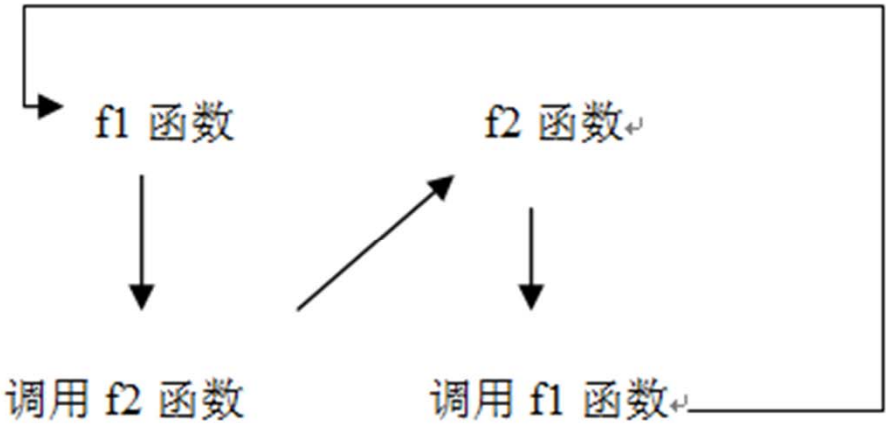


图 6.8　间接调用

6.6 函数的递归调用

为了防止递归调用无终止地进行，必须在函数内有终止递归调用的手段。常用的办法是加条件判断，满足某种条件后就不再作递归调用，然后逐层返回。

一个递归的过程可以分为“回推”和“递推”两个阶段。第一阶段是“回推”，即将未知逐层回推到已知，然后开始第二阶段，采用递推方法，从已知推算出未知，一直推算出问题解为止。下面用一个通俗的例子来说明。

例6.8 有6个坐在一桌，问第6个人多少岁？，他说比第5个人大2岁。问第5个人多少岁？他说比第4个人大2岁。问第4个人多少岁？他说比第3个人大2岁。问第3个人多少岁？他说比第2个人大2岁。问第2个人多少岁？他说比第1个人大2岁。最后问第1个人，他说他6岁。请问第6个人多大？

6.6 函数的递归调用

显然，这是一个递归问题。依题意：

age(6)=age(5)+2
age(5)=age(4)+2
age(4)=age(3)+2
age(3)=age(2)+2
age(2)=age(1)+2
age(1)=6

用数学公式来表示：

$$age(n)=\begin{cases}6 & (n=1)\\ age(n-1)+2 & (n>1)\end{cases}$$

用递归解决此问题的程序如下：

```
#include<stdio.h>
int age(int a)
{ int c;
  if( n = = 1)
    c=6;
  else
    c=age(n-1)+2;
return(c);
}
void main()
{
  printf("%d\n",age(6));
 }
```

6.6 函数的递归调用

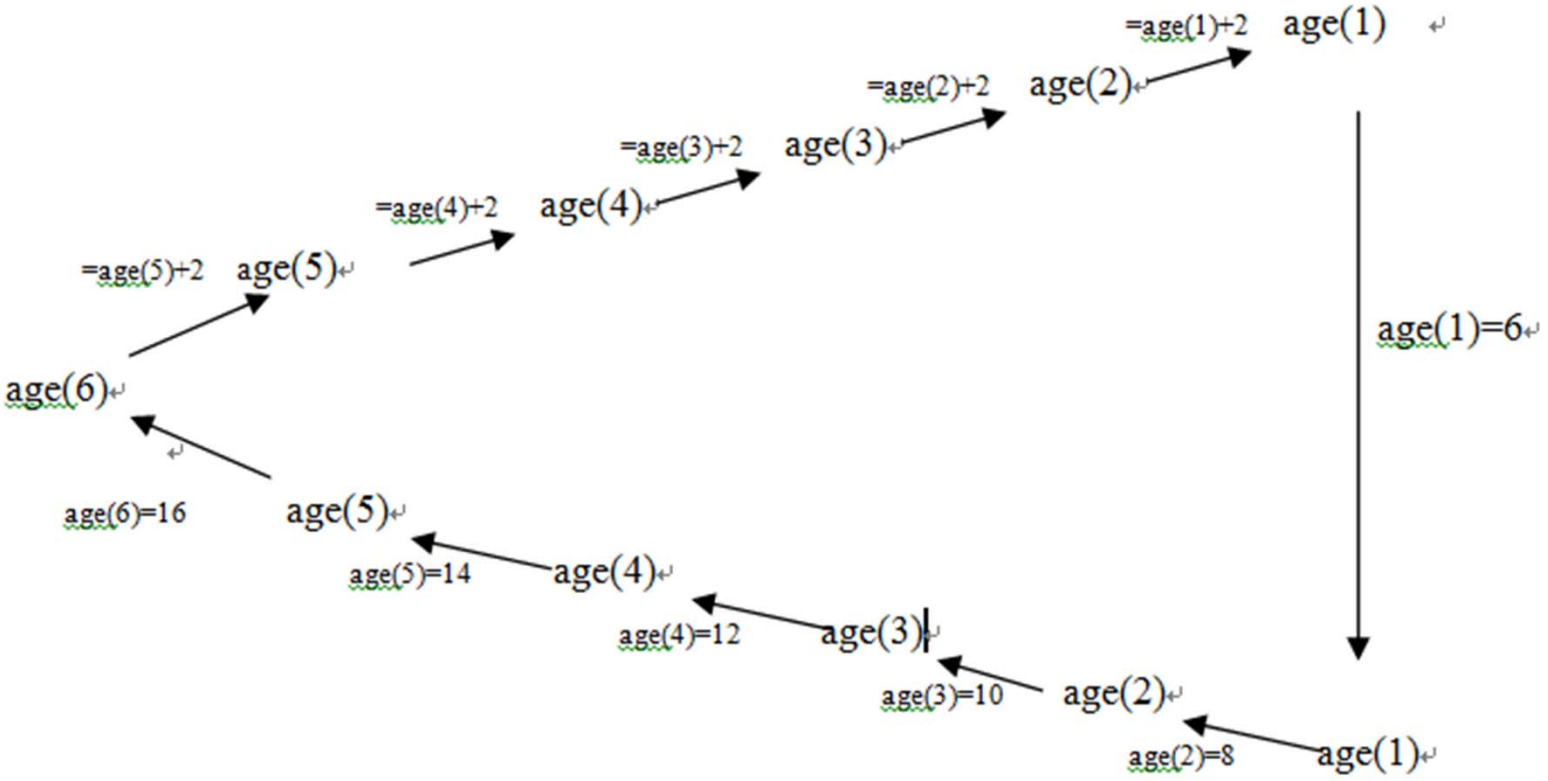


图 6.10　递归的求解过称

6.6 函数的递归调用

例6.9 用递归法计算n!

用递归法计算n!, 可用下述公式表示：

```
n!=1          (n=0,1)
n×(n-1)!      (n>1)
```

按公式可编程如下：

```
#include <stdio.h>
long ff(int n){
    long f;
    if(n<0) printf("n<0,input error");
    else if(n==0||n==1) f=1;
    else f=ff(n-1)*n;
    return(f);
}
void main(){
    int n;
    long y;
    printf("\n input a inteager number:\n");
    scanf("%d",&n);
    y=ff(n);
    printf("%d!=%ld",n,y);
}
```

程序中给出的函数ff是一个递归函数。主函数调用ff 后即进入函数ff执行，如果n<0,n==0或n=1时都将结束函数的执行，否则就递归调用ff函数自身。

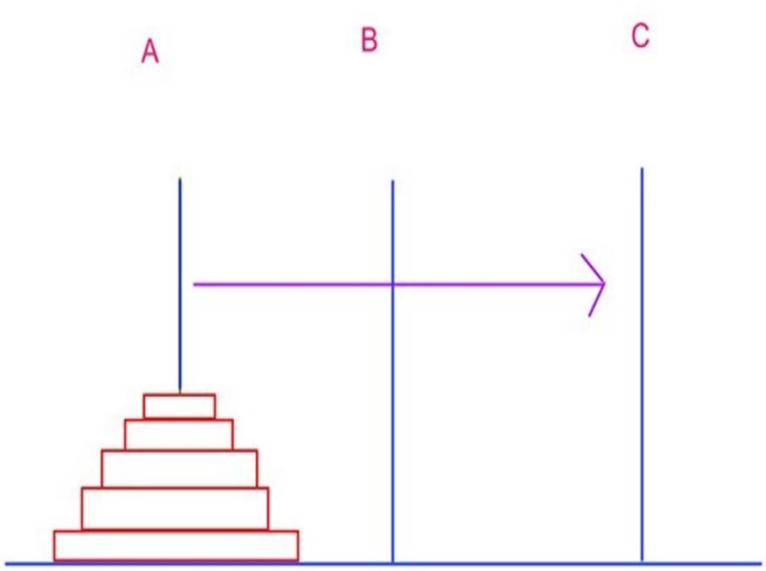
由于每次递归调用的实参为n-1，即把n-1的值赋予形参n，最后当n-1的值为1时再作递归调用，形参n的值也为1，将使递归终止。然后可逐层退回。

6.6 函数的递归调用

例6.10 Hanoi塔问题

汉诺塔问题是现代递归算法思想的来源，是一个很经典的问题。汉诺塔问题是印度的一个古老的传说。开天辟地的神勃拉玛在一个庙里留下了三根金刚石的棒，第一根上面套着64个圆的金片，最大的一个在底下，其余一个比一个小，依次叠上去，庙里的众僧不倦地把它们一个个地从这根棒搬到另一根棒上，规定可利用中间的一根棒作为帮助，但每次只能搬一个，而且大的不能放在小的上面。为方便，我们给这三根棒命名为A，B，C。A棒上套有64个大小不等的圆盘，大的在下，小的在上。如图6.11所示。要把这64个圆盘从A棒移动C棒上，每次只能移动一个圆盘，移动可以借助B棒进行。但在任何时候，任何棒上的圆盘都必须保持大盘在下，小盘在上。求移动的步骤。

6.6 函数的递归调用



本题算法分析如下，设A上有n个盘子。
如果n=1，则将圆盘从A直接移动到C。
如果n=2，则：
1.将A上的n-1(等于1)个圆盘移到B上；
2.再将A上的一个圆盘移到C上；
3.最后将B上的n-1(等于1)个圆盘移到C上。
如果n=3，则：
A. 将A上的n-1(等于2，令其为n`)个圆盘移到B(借助于C)，步骤如下：
(1)将A上的n`-1(等于1)个圆盘移到C上。
(2)将A上的一个圆盘移到B。
(3)将C上的n`-1(等于1)个圆盘移到B。
B. 将A上的一个圆盘移到C。
C. 将B上的n-1(等于2，令其为n`)个圆盘移到C(借助A)，步骤如下：
(1)将B上的n`-1(等于1)个圆盘移到A。
(2)将B上的一个盘子移到C。
(3)将A上的n`-1(等于1)个圆盘移到C。

6.6 函数的递归调用

到此，完成了三个圆盘的移动过程。

从上面分析可以看出，当n大于等于2时，移动的过程可分解为三个步骤：

第一步 把A上的n-1个圆盘移到B上；

第二步 把A上的一个圆盘移到C上；

第三步 把B上的n-1个圆盘移到C上；其中第一步和第三步是类同的。

当n=3时，第一步和第三步又分解为类同的三步，即把n`-1个圆盘从一个棒移到另一个棒上，这里的n`=n-1。 显然这是一个递归过程

6.6 函数的递归调用

```
#include <stdio.h>
void move(int n,int x,int y,int z)
{
 if(n==1)
       printf("%c-->%c\n",x,z);
    else
    {
      move(n-1,x,z,y);
      printf("%c-->%c\n",x,z);
      move(n-1,y,x,z);
    }
}
void main()
{
    int h;
    printf("\ninput number:\n");
    scanf("%d",&h);
    printf("the step to moving %2d
diskes:\n",h);
    move(h,'a','b','c');
}
```

move函数是一个递归函数，它有四个形参n,x,y,z。n表示圆盘数，x,y,z分别表示三根棒。move 函数的功能是把x上的n个圆盘移动到z上。

如n!=1则分为三步：递归调用move函数，把n-1个圆盘从x移到y；输出x→z；递归调用move函数，把n-1个圆盘从y移到z。在递归调用过程中n=n-1，故n的值逐次递减，最后n=1时，终止递归，逐层返回。

第六章 函数

6.7 局部变量和全局变量

在讨论函数的形参变量时曾经提到，形参变量只在被调用期间才分配内存单元，调用结束立即释放。这一点表明形参变量只有在函数内才是有效的，离开该函数就不能再使用了。这种变量有效性的范围称变量的作用域。不仅对于形参变量，C语言中所有的量都有自己的作用域。变量说明的方式不同，其作用域也不同。C语言中的变量，按作用域范围可分为两种，即局部变量和全局变量。

6.7 局部变量和全局变量

在一个函数内部定义的变量是内部变量，内部变量只在本函数范围的内部有效，只能在本函数的内部使用它们，在此函数之外就不能使用这些变量了。所以内部变量也称“局部变量”。

```
int f1(int a)                    /*函数 f1*/
  { int b,c;
      ……
  }                              /*a,b,c 作用域：仅限于函数 f1()中*/

int f2(int x)                    /*函数 f2*/
  { int y,z;
      ……
  }                              /*x,y,z 作用域：仅限于函数 f2()中*/

void main()
    { int m,n;
        ……
    }                            /*m,n 作用域：仅限于函数 main()中*/
```

6.7 局部变量和全局变量

关于局部变量的作用域说明以下几点：

⑴ 主函数main()中定义的局部变量，也只能在主函数中使用，其它函数不能使用。同时，主函数中也不能使用其它函数中定义的局部变量。因为主函数也是一个函数，与其它函数是平行关系。这一点是与其它语言不同的，应予以注意。

⑵ 形参变量也是局部变量，属于被调用函数；实参变量，则是调用函数的局部变量。

⑶ 允许在不同的函数中使用相同的变量名，它们代表不同的对象，分配不同的单元，互不干扰，也不会发生混淆。

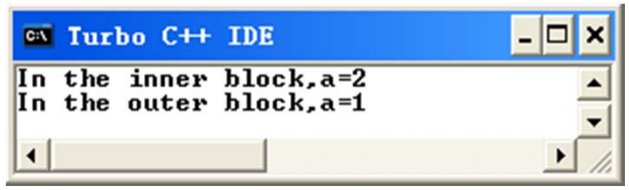
⑷ 在一个函数内部，在复合语句中也可以定义变量，这些变量只在本复合语句中有效。

6.7 局部变量和全局变量

例6.11

```
#include <stdio.h>
void main()
{int  a=1;
    {int a=2;
  printf("In the inner block,a=%d\n",a);
      }
  printf ("In the outer block,a=%d\n",a);
}
```

变量的作用域规则是：每个变量仅在定义它的复合语句（包含下级复合语句）内有效，并且拥有自己的内存空间。



可见，在内层复合语句使用的变量是它自己的a，而非外层定义的a，注意，这两个a是真正的两个a，各有自己的存储空间，彼此毫无关系。

6.7 局部变量和全局变量

6.7.2 全局变量

全局变量也称为外部变量，它是在函数外部定义的变量。它不属于哪一个函数，它属于一个源程序文件。其作用域是整个源程序。在函数中使用全局变量，一般应作全局变量说明。 只有在函数内经过说明的全局变量才能使用。

如下：

6.7 局部变量和全局变量

例如：

```
int a=1,b=2;                /*外部变量*/
float f1(int x)             /*定义函数 f1*/
   {
      int c;
      …
   }

char   c1,c2;               /*外部变量*/
char f2(int y)              /*定义函数 f2*/
   {
    int z;
    …
   }

void   main()               /*主函数*/
   {
      int m;
      float f;
      …
   }
```

全局变量 c1,c2 的作用范围

全局变量 a、b 的作用范围

6.7 局部变量和全局变量

对于全局变量的几点说明：

⑴ 全局变量可加强函数模块之间的数据联系，但又使这些函数依赖这些全局变量，如果在一个函数中改变了全局变量的值，就能影响到其他函数，因而使得这些函数的独立性降低。从模块化程序设计的观点来看这是不利的，因此不是非用不可时，不要使用外部变量。

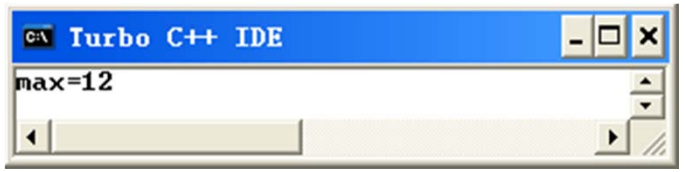
⑵ 在同一源文件中，允许全局变量和局部变量同名。在局部变量的作用域内，全局变量将被屏蔽而不起作用。

⑶ 全局变量的作用域是从定义点到本文件结束。

6.7 局部变量和全局变量

例6.12　外部变量与局部变量同名。

```
#include <stdio.h>
int a=5,b=8;              /*定义全局变量*/
int max(int a,int b)          /*定义函数，其中a,b是形参*/
{                           /*形参a,b是局部变量与全局变量同名*/
  int c;
  c=a>b?a:b;
  return(c);
}
void main()
{
  int a=12;                 /*局部变量a与全局变量同名*/
  printf("max=%d",max(a,b));
}
```



第六章 函数

6.8 存储变量的类别

6.8.1 动态存储方式与静态存储方式

从变量的作用域（即从空间）角度来分，可以分为全局变量和局部变量。

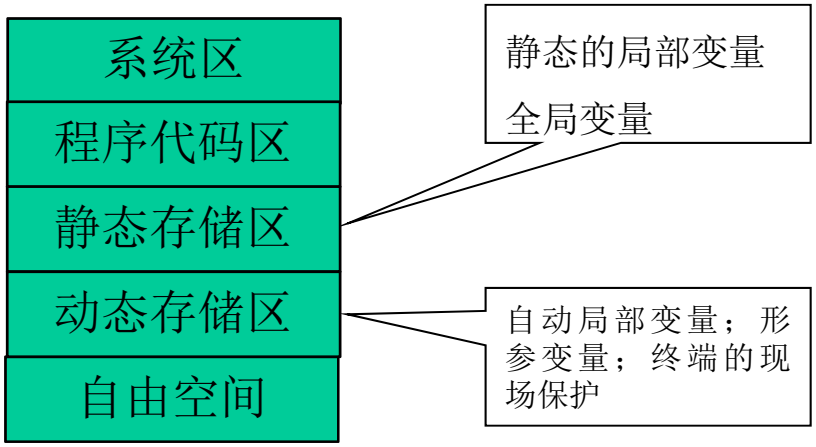
从变量值存在的作时间（即生存期）角度来分，可以分为静态存储方式和动态存储方式。

静态存储方式：是指在程序运行期间分配固定的存储空间的方式。

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存储空间的方式。

程序运行时的内存分配情况，如右图：

1. 系统区
2. 程序代码区
3. 静态存储区
4. 动态存储区
5. 自由空间



6.8 存储变量的类别

6.8.2 auto变量

6.8 存储变量的类别

6.8.3　用statice声明局部变量

6.8 存储变量的类别

6.8.4 register变量

6.8 存储变量的类别

6.8.4　extern声明外部变量

6.8 存储变量的类别

第六章 函数

6.9 习题