## 第四章 函数

#### 一、函数

##### 1.函数的概念

“一个程序应该是轻灵自由的，它的子过程就像串在一根线上的珍珠”，子过程被称为“函数”，程序的执行从main的入口开始，到main的出口结束，中间往复、循环、迭代地调用一个又一个函数。

从用户角度，函数可以分为：

* 标准函数，即库函数
* 用户自定义的函数

从函数形式可以分为:

* 无参函数
* 有参函数

##### 2.函数的定义

函数的定义语法如下：

返回值类型 函数名（类型 参数1，类型 参数2，……）

{

局部变量定义语句

可执行语句序列

}

##### 3.函数的参数

函数的参数：用于函数间数据的传递

1.通过名称理解

形式参数：形式上存在的参数

实际参数：实际存在的参数

2.通过作用理解

形式参数：定义函数时使用的参数。在函数调用前，传递给函数的值将被复制到这些形式参数中。

实际参数：调用时传递给函数的参数

#include <stdio.h>

Int max(int x,int y)

{

Int z;

z=(x>y)?x:y;

return (z);

}

void main()

{

int a,b,c;

scanf(“%d%d”,&a,&b);

c=max(a,b);

printf(“max is %d\n”,c);

}

对于上面的例子，x,y是函数max的两个形参，a,b是主函数中定义的变量，当main函数调用max函数时，a,b成为max的实际参数，调用时，实际参数的值赋给形参进行处理

定义函数时，必须说明形参的类型，形参只能是变量，不能是常量或表达式。

函数被调用之前，形参和子函数中的变量不占内存，调用结束返回，形参所占的内存被回收。

实参可以是常量、变量或表达式，因为传递过来的是具体数值。

实参和形参类型必须一致（或可以安全转换）。

C语言中，实参和形参传递的是“按值传递”，即单向传递，只与参数相对位置有关，而与变量名无关。

##### 4.函数的返回值

函数值也就是函数的返回值，是一个具体确定的值。

函数使用return 语句返回值，一个函数内可以有多个return语句，执行到任何return语句，函数都将立刻返回到调用函数。

return后面的（）可以省略，可以返回一个表达式，先求表达式的值，再返回。

如下面的例子：

#include <stdio.h>

int max(int x,int y)

{

If(x>y)

return (x);

else

return (y);

}

int max(int x,int y)

{

return x>y?x:y;

}

函数的类型即函数值的类型，例如函数max是int型，函数的返回值也是int型。

省略了类型说明的函数就是int型。

return中表达式的值一般和函数类型相同。如果不一致，则需要进行类型转换，

以函数类型为准。

不需要返回的函数

如果函数中没有return,认为是一个过程，可以使用类型void。

如果一个函数被声明void类型，就不允许再引用它的返回值，只能单纯调用它。

#### 二、变量的作用域和存储类型

C语言中所有变量都有自己的作用域，申明变量的类型不同，其作用域也不同。C语言中的变量，按照作用域的范围可分为两种， 即局部变量和全局变量。

1.局部变量

局部变量也称为内部变量。局部变量是在函数内作定义说明的。其作用域仅限于函数内，离开该函数后再使用这种变量是非法的。

例如：

int f1(int a) /\*函数f1\*/

{

    int b,c；

……

}

f1内定义了三个变量，a为形参，b,c为一般变量。在f1的范围内a,b,c有效，或者说a,b,c变量的作用域限于f1内。

int f2(int x) /\*函数f2\*/

{

    int y,z;；

……

}

f2内定义了三个变量，x为形参，y,z为一般变量。在 f2的范围内x,y,z有效，或者说x,y,z变量的作用域限于f2内。

关于局部变量的作用域还要说明以下几点：

主函数中定义的变量也只能在主函数中使用，不能在其它函数中使用。同时，主函数中也不能使用其它函数中定义的变量。因为主函数也是一个函数，它与其它函数是平行关系。

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。虽然允许在不同的函数中使用相同的变量名，但是为了使程序明了易懂，不提倡在不同的函数中使用相同的变量名。

##### 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。全局变量的说明符为extern。但在一个函数之前定义的全局变量，在该函数内使用可不再加以说明。

例如：

float x,y; /\*外部变量\*/

int fz() /\*函数fz\*/

{

……

}

int a,b; /\*外部变量\*/

void f1() /\*函数f1\*/

{

……

}

　从上例可以看出a、b、x、y 都是在函数外部定义的外部变量，都是全局变量。

　　对于全局变量还有以下几点说明：

  对于局部变量的定义和说明，可以不加区分。而对于外部变量则不然，外部变量的定义和外部变量的说明并不是一回事。外部变量定义必须在所有的函数之外，且只能定义一次。其一般形式为：

 [extern] 类型说明符 变量名，变量名…

其中方括号内的extern可以省去不写。

　　例如： int a,b;

　　等效于：

  extern int a,b;

　　而外部变量说明出现在要使用该外部变量的各个函数内，在整个程序内，可能出现多次，外部变量说明的一般形式为：

extern 类型说明符 变量名，变量名，…；

外部变量在定义时就已分配了内存单元，外部变量定义可作初始赋值，外部变量说明不能再赋初始值，只是表明在函数内要使用某外部变量。

外部变量可加强函数模块之间的数据联系，但是又使函数要依赖这些变量，因而使得函数的独立性降低。从模块化程序设计的观点来看这是不利的， 因此在不必要时尽量不要使用全局变量。

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，全局变量不起作用。

int vs(int l,int w)

{

extern int h;

int v;

v=l\*w\*h;

return v;

}

main()

{

extern int w,h;

int l=5;

printf("v=%d",vs(l,w));

}

int l=3,w=4,h=5;

本例程序中,外部变量在最后定义,因此在前面函数中对要用的外部变量必须进行说明。外部变量l，w和vs函数的形参l，w同名。外部变量都作了初始赋值，main函数中也对l作了初始化赋值。执行程序时，在printf语句中调用vs函数，实参l的值应为main中定义的l值，等于5，外部变量l在main内不起作用；实参w的值为外部变量w的值为4，进入vs后这两个值传送给形参l，w，vs函数中使用的h 为外部变量，其值为5，因此v的计算结果为100，返回主函数后输出。

变量的存储类型决定了各种变量的作用域不同。所谓存储类型是指变量占用内存空间的方式，也称为存储方式。变量的存储方式可分为“静态存储”和“动态存储”两种。

静态存储变量通常是在变量定义时就分定存储单元并一直保持不变,直至整个程序结束。动态存储变量是在程序执行过程中，使用它时才分配存储单元，使用完毕立即释放。

典型的例子是函数的形式参数，在函数定义时并不给形参分配存储单元，只是在函数被调用时，才予以分配，调用函数完毕立即释放。

如果一个函数被多次调用，则反复地分配、释放形参变量的存储单元。从以上分析可知，静态存储变量是一直存在的，而动态存储变量则时而存在时而消失。我们又把这种由于变量存储方式不同而产生的特性称变量的生存期。生存期表示了变量存在的时间。

生存期和作用域是从时间和空间这两个不同的角度来描述变量的特性，这两者既有联系，又有区别。一个变量究竟属于哪一种存储方式，并不能仅从其作用域来判断，还应有明确的存储类型说明。

在C语言中，对变量的存储类型说明有以下四种：

auto　　　　  自动变量

　　register 　　 寄存器变量

　　extern 　　　外部变量

　　static 　　　静态变量

　　自动变量和寄存器变量属于动态存储方式，外部变量和静态变量属于静态存储方式。在介绍了变量的存储类型之后，可以知道对一个变量的说明不仅应说明其数据类型，还应说明其存储类型。因此变量说明的完整形式为：

 存储类型说明符 数据类型说明符 变量名，变量名…；

例如：

　　 static int a,b; 　　　　　　　　　 说明a,b为静态类型变量

　　 auto char c1,c2; 　　　　　　　　　说明c1,c2为自动字符变量

　　 static int a[5]={1,2,3,4,5}; 　　　说明a为静态整型数组

　　 extern int x,y; 　　　　　　　　　 说明x,y为外部整型变量

　　下面分别介绍以上四种存储类型：

　　1）自动变量的类型说明符为auto

　这种存储类型是C语言程序中使用最广泛的一种类型。C语规定，函数内凡未加存储类型说明的变量均视为自动变量，也就是说自动变量可省去说明符auto。

例如：

{ int i,j,k;

char c;

……

}等价于：

{ auto int i,j,k;

auto char c;

……

}

　　自动变量具有以下特点：

　　自动变量的作用域仅限于定义该变量的个体内。在函数中定义的自动变量，只在该函数内有效。在复合语句中定义的自动变量只在该复合语句中有效。

例如：

int kv(int a)

{

auto int x,y;

{

auto char c;

} /\*c的作用域\*/

……

} /\*a,x,y的作用域\*/

　　自动变量属于动态存储方式，只有在使用它，即定义该变量的函数被调用时才给它分配存储单元，开始它的生存期。函数调用结束，释放存储单元，结束生存期。因此函数调用结束之后，自动变量的值不能保留。在复合语句中定义的自动变量，在退出复合语句后也不能再使用，否则将引起错误。例如以下程序：

main()

{

auto int a,s,p;

printf("\ninput a number:\n");

scanf("%d",&a);

if(a>0)

{

s=a+a;

p=a\*a;

}

printf("s=%d p=%d\n",s,p);

}

{

auto int a;

printf("\ninput a number:\n");

scanf("%d",&a);

if(a>0)

{

auto int s,p;

s=a+a;

p=a\*a;

}

printf("s=%d p=%d\n",s,p);

}

　　s,p是在复合语句内定义的自动变量，只能在该复合语句内有效。而程序退出复合语句之后用printf语句输出s,p的值，这显然会引起错误。

　 由于自动变量的作用域和生存期都局限于定义它的个体内(函数或复合语句内)，因此不同的个体中允许使用同名的变量而不会混淆。即使在函数内定义的自动变量也可与该函数内部的复合语句中定义的自动变量同名。

main()

{

auto int a,s=100,p=100;

printf("/ninput a number:/n");

scanf("%d",&a);

if(a>0)

{

auto int s,p;

s=a+a;

p=a\*a;

printf("s=%d p=%d/n",s,p);

}

printf("s=%d p=%d/n",s,p);

}

　　本程序在main函数中和复合语句内两次定义了变量s,p为自动变量。按照C语言的规定，在复合语句内，应由复合语句中定义的s,p起作用，故s的值应为a+ a，p的值为a\*a。退出复合语句后的s,p应为main所定义的s,p，其值在初始化时给定，均为100。从输出结果可以分析出两个s和两个p虽变量名相同，但却是两个不同的变量。

　 对构造类型的自动变量如数组等，不可作初始化赋值。

###### 2) 外部变量的类型说明符为extern

　　在前面介绍全局变量时已介绍过外部变量。这里再补充说明外部变量的几个特点：

　　 1. 外部变量和全局变量是对同一类变量的两种不同角度的提法。全局变量是从它的作用域提出的，外部变量从它的存储方式提出的，表示了它的生存期。

　　 2. 当一个源程序由若干个源文件组成时，在一个源文件中定义的外部变量在其它的源文件中也有效。例如有一个源程序由源文件F1.C和F2.C组成：

F1.C

int a,b; /\*外部变量定义\*/

char c; /\*外部变量定义\*/

main()

{

……

}

F2.C

extern int a,b; /\*外部变量说明\*/

extern char c; /\*外部变量说明\*/

func (int x,y)

{

……

}

　在F1.C和F2.C两个文件中都要使用a,b,c三个变量。在F1.C文件中把a,b,c都定义为外部变量。在F2.C文件中用extern把三个变量说明为外部变量，表示这些变量已在其它文件中定义，并把这些变量的类型和变量名，编译系统不再为它们分配内存空间。

对构造类型的外部变量，如数组等可以在说明时作初始化赋值，若不赋初值，则系统自动定义它们的初值为0。

静态变量

　静态变量的类型说明符是static。 静态变量当然是属于静态存储方式，但是属于静态存储方式的量不一定就是静态变量，例如外部变量虽属于静态存储方式，但不一定是静态变量，必须由static加以定义后才能成为静态外部变量，或称静态全局变量。

对于自动变量，前面已经介绍它属于动态存储方式。但是也可以用static定义它为静态自动变量，或称静态局部变量，从而成为静态存储方式。

由此看来，一个变量可由static进行再说明，并改变其原有的存储方式。

###### 3）静态局部变量

　在局部变量的说明前再加上static说明符就构成静态局部变量。

　例如：

static int a,b;

static float array[5]={1,2,3,4,5}；

　静态局部变量属于静态存储方式，它具有以下特点：

　(1)静态局部变量在函数内定义，但不象自动变量那样，当调用时就存在，退出函数时就消失。静态局部变量始终存在着，也就是说它的生存期为整个源程序。

　(2)静态局部变量的生存期虽然为整个源程序，但是其作用域仍与自动变量相同，即只能在定义该变量的函数内使用该变量。退出该函数后， 尽管该变量还继续存在，但不能使用它。

　(3)允许对构造类静态局部量赋初值。在数组一章中，介绍数组初始化时已作过说明。若未赋以初值，则由系统自动赋以0值。

　(4)对基本类型的静态局部变量若在说明时未赋以初值，则系统自动赋予0值。而对自动变量不赋初值，则其值是不定的。 根据静态局部变量的特点， 可以看出它是一种生存期为整个源程序的量。虽然离开定义它的函数后不能使用，但如再次调用定义它的函数时，它又可继续使用，而且保存了前次被调用后留下的值。 因此，当多次调用一个函数且要求在调用之间保留某些变量的值时，可考虑采用静态局部变量。虽然用全局变量也可以达到上述目的，但全局变量有时会造成意外的副作用，因此仍以采用局部静态变量为宜。

main()

{

int i;

void f(); /\*函数说明\*/

for(i=1;i<=5;i++)

f(); /\*函数调用\*/

}

void f() /\*函数定义\*/

{

auto int j=0;

++j;

printf("%d/n",j);

}

　　程序中定义了函数f，其中的变量j 说明为自动变量并赋予初始值为0。当main中多次调用f时，j均赋初值为0，故每次输出值均为1。现在把j改为静态局部变量，程序如下：

main()

{

int i;

void f();

for (i=1;i<=5;i++)

f();

}

void f()

{

static int j=0;

++j;

printf("%d/n",j);

}

void f()

{

static int j=0;

++j;

printf("%d/n",j);

}

　　由于j为静态变量，能在每次调用后保留其值并在下一次调用时继续使用，所以输出值成为累加的结果。读者可自行分析其执行过程。

###### 4）静态全局变量

　　 全局变量(外部变量)的说明之前再冠以static 就构成了静态的全局变量。全局变量本身就是静态存储方式，静态全局变量当然也是静态存储方式。这两者在存储方式上并无不同。这两者的区别虽在于非静态全局变量的作用域是整个源程序，当一个源程序由多个源文件组成时，非静态的全局变量在各个源文件中都是有效的。而静态全局变量则限制了其作用域，即只在定义该变量的源文件内有效，在同一源程序的其它源文件中不能使用它。由于静态全局变量的作用域局限于一个源文件内，只能为该源文件内的函数公用，因此可以避免在其它源文件中引起错误。从以上分析可以看出，把局部变量改变为静态变量后是改变了它的存储方式即改变了它的生存期。把全局变量改变为静态变量后是改变了它的作用域，限制了它的使用范围。因此static这个说明符在不同的地方所起的作用是不同的。

##### 3.寄存器变量

上述各类变量都存放在存储器内，因此当对一个变量频繁读写时，必须要反复访问内存储器，从而花费大量的存取时间。为此，C语言提供了另一种变量，即寄存器变量。这种变量存放在CPU的寄存器中，使用时，不需要访问内存，而直接从寄存器中读写，这样可提高效率。寄存器变量的说明符是register。 对于循环次数较多的循环控制变量及循环体内反复使用的变量均可定义为寄存器变量。

求∑200i=1imain()

{

register i,s=0;

for(i=1;i<=200;i++)

s=s+i;

printf("s=%d/n",s);

}

　本程序循环200次，i和s都将频繁使用，因此可定义为寄存器变量。对寄存器变量还要说明以下几点：

　1. 只有局部自动变量和形式参数才可以定义为寄存器变量。因为寄存器变量属于动态存储方式。凡需要采用静态存储方式的量不能定义为寄存器变量。

　2. 在Turbo C，MS C等微机上使用的C语言中，实际上是把寄存器变量当成自动变量处理的。因此速度并不能提高。 而在程序中允许使用寄存器变量只是为了与标准C保持一致。3. 即使能真正使用寄存器变量的机器，由于CPU中寄存器的个数是有限的，因此使用寄存器变量的个数也是有限的。

#### 三、函数递归

##### 1.什么是递归

递归函数即自调用函数，在函数体内部直接或间接地自己调用自己，即函数的嵌套调用是函数本身。

  　　例如，下面的程序为求n!：

  　　　　long fact(int n)

  　　　　{

  　　　　　if(n==1)

  　　　　　return 1；

  　　　　　return fact(n-1)\*n； ／／出现函数自调用

  　　　　}

##### 2.递归调用机制的说明

任何函数之间不能嵌套定义,调用函数与被调用函数之间相互独立(彼此可以调用)。发生函数调用时，被调函数中保护了调用函数的运行环境和返回地址，使得调用函数的状态可以在被调函数运行返回后完全恢复，而且该状态与被调函数无关。

  被调函数运行的代码虽是同一个函数的代码体，但由于调用点，调用时状态，返回点的不同，可以看作是函数的一个副本，与调用函数的代码无关，所以函数的代码是独立的。被调函数运行的栈空间独立于调用函数的栈空间，所以与调用函数之间的数据也是无关的。函数之间靠参数传递和返回值来联系，函数看作为黑盒。这种机制决定了C／C++允许函数递归调用。

##### 3.递归调用的形式

　 递归调用有直接递归调用和间接递归调用两种形式。

  直接递归即在函数中出现调用函数本身。

  例如，下面的代码求斐波那契数列第n项。斐波那契数列的第一和第二项是1，后面每一项是前二项之和，即1，1，2，3，5，8，13,...。 代码中采用直接递归调用：

  　　　　long fib(int x)

  　　　　{

  　　　　　if(x>2)

  　　　　　　return(fib(x-1)+fib(x-2))； //直接递归

  　　　　　else

  　　　　　　return 1；

  　　　　}

  　　间接递归调用是指函数中调用了其他函数，而该其他函数却又调用了本函数。例如，下面的代码定义两个函数，它们构成了间接递归调用：

  　　　　int fnl(int a)

  　　　　{

  　　　　　int b；

  　　　　　b=fn2(a+1)； //间接递归

  　　　　　　　　　　　//...

  　　　　}

  　　　　int fn2(int s)

  　　　　{

  　　　　　int c；

  　　　　　c=fnl(s-1); //间接递归

  　　　　　　　　　　　//...

  　　　　}

  上例中，fn1()函数调用了fn2()函数，而fn2()函数又调用了fn1()函数。

##### 4.递归的条件

(1)须有完成函数任务的语句。

  例如，下面的代码定义了一个递归函数：

  　　　　#include

  　　　　void count(int val) //递归函数可以没有返回值

  　　　　{ if(val>1)

  　　　　　　　count(val-1)；

  　　　　　printf(“ok\n”);" 此语句完成函数任务=""

}

  该函数的任务是在输出设备上显示"ok：整数值”。

  (2)—个确定是否能避免递归调用的测试

  例如，上例的代码中，语句"if(val>1)"便是—个测试，如果不满足条件，就不进行递归调用。

  (3)一个递归调用语句。

  该递归调用语句的参数应该逐渐逼近不满足条件，以至最后断绝递归。

  例如，上面的代码中，语句“if(val>1)” 便是一个递归调用，参数在渐渐变小，这种发展趋势能使测试"if(val>1)”最终不满足。

  (4)先测试，后递归调用。

  在递归函数定义中，必须先测试，后递归调用。也就是说，递归调用是有条件的，满足了条件后，才可以递归。

 　例如，下面的代码无条件调用函数自己，造成无限制递归，终将使栈空间溢出：

  　　　　#include

  　　　　void count(int val)

  　　　　{

  　　　　　count(val-1)； //无限制递归

  　　　　　if(val>1) //该语句无法到达

  　　　　　　cout <<"ok： " <<　　　　}

##### 5．递归的评价

递归的目的是简化程序设计，使程序易读。但递归增加了系统开销。时间上，执行调用与返回的额外工作要占用CPU时间。空间上，随着每递归一次，栈内存就多占用一截。相应的非递归函数虽然效率高，但却比较难编程，而且相对来说可读性差。

  现代程序设计的目标主要是可读性好。随着计算机硬件性能的不断提高，程序在更多的场合优先考虑可读而不是高效，所以，鼓励用递归函数实现程序思想。

#### 任务

##### 任务一：双曲正弦函数

输入x的值，计算y的值。已知y=(sh(1+shx))/(sh2x+sh3x)

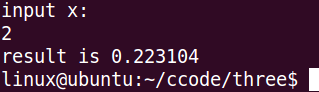
其中shx是双曲正弦函数：

sh x = (e^x - e^(-x)) / 2

运行：

输入：2

结果：



#include <stdio.h>

#include <math.h>

double f(double x)

{

    return (exp(x) - exp(-x)) / 2;

}

int main()

{

    printf("input x:\n");

    double x = 0;

    scanf("%lf", &x);

    printf("result is %.6f", f(1 + f(x)) / (f(2 \* x) + f(3 \* x)));

    return 0;

}

##### 任务二：求素数

编写程序输出100以内的素数。

分析：可以编写一函数来判断一个数m是否为素数。如果返回1，则是素数；如果返回0，则不是素数。

#include <stdio.h>

int is\_Prime(int n)

{

    if (n <= 1)

        return 0;

    for (int i = 2; i \* i <= n; i++)

    {

        if (n % i == 0)

        {

            return 0;

        }

    }

    return 1;

}

int main()

{

    for (int i = 1; i <= 100; i++)

    {

        if (is\_Prime(i))

        {

            printf("%d\n", i);

        }

    }

    return 0;

}

##### 任务三： 求阶乘之和

请用自定义函数的形式编程实现，求S=m!+n!+k!,m n k从键盘输入（值均小于7）

#include <stdio.h>

int f(int x)

{

    int sum = 1;

    int i = 0;

    for (i = x; i >= 1; i--)

    {

        sum \*= i;

    }

    return sum;

}

int main()

{

    int m, n, k;

    scanf("%d %d %d", &m, &n, &k);

    int S = f(m) + f(n) + f(k);

    printf("S = %d\n", S);

    return 0;

}

##### 任务四：歌星大奖赛

在歌星大奖赛中，有10个评委为参赛的选手打分，分数为1〜100分。选手最后得分为：去掉一个最高分和一个最低分后其余8个分数的平均值。请编写一个程序实现。

#include <stdio.h>

int main()

{

    int a[10], mx, mn;

    double sum = 0;

    int i;

    for (i = 0; i < 10; i++)

    {

        scanf("%d", &a[i]);

        sum += a[i];

    }

    mx = a[0];

    mn = a[0];

    for (i = 1; i < 10; i++)

    {

        if (mx < a[i])

        {

            mx = a[i];

        }

        if (mn > a[i])

        {

            mn = a[i];

        }

    }

    sum = (sum - mn - mx) \* 1.0 / 8;

    printf("sum == %.2f", sum);

    return 0;

}

##### 任务五：找兔子

一只兔子躲进了10个环形分布的洞中的一个。狼在第一个洞中没有找到兔子，就隔一个洞，到第3个洞去找；也没有找到，就隔2个洞，到第6个洞去找；以后每次多一个洞去找兔子……这样下去，如果一直找不到兔子，请问兔子可能在哪个洞中？

问题分析：

首先定义一个数组a[11]，其数组元素为a[1]，a[2]，a[3]……a[10]，这10个数组元素分别表示10个洞，初值均置为1。

接着使用“穷举法”来找兔子，通过循环结构进行穷举，设最大寻找次数为1000次。由于洞只有10个，因此第n次查找对应第n%10个洞，如果在第n%10个洞中没有找到兔子，则将数组元素a[n%10]置0。

当循环结束后，再检查a数组各元素（各个洞）的值，若其值仍为1，则兔子可能藏身于该洞中。

#include <stdio.h>

int main()

{

    int a[11];

    memset(a, 1, sizeof a);

    int Max;

    scanf("%d", &Max);

    int cnt = 1;

    int cur = 0;

    while (Max--)

    {

        a[cur % 10] = 0;

        cur += cnt + 1;

        cnt++;

    }

    int i;

    for (i = 0; i < 10; i++)

    {

        if (a[i] == 1)

        {

            printf("有未找到的兔子\n");

        }

    }

    return 0;

}

##### 任务六:魔方阵

编写程序，实现如下所示的5-魔方阵：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 17 | 24 | 1 | 8 | 15 |
| 23 | 5 | 7 | 14 | 16 |
| 4 | 6 | 13 | 20 | 22 |
| 10 | 12 | 19 | 21 | 3 |
| 11 | 18 | 25 | 2 | 9 |

问题分析:

所谓“n-魔方阵”，指的是使用1〜n2共n2个自然数排列成一个n×n的方阵，其中n为奇数；该方阵的每行、每列及对角线元素之和都相等，并为一个只与n有关的常数，该常数为n×(n^2+1)/2。

例如5-魔方阵，其第一行、第一列及主对角线上各元素之和如下：

* 第一行元素之和：17+24+1+8+15=65
* 第一列元素之和：17+23+4+10+11=65
* 主对角线上元素之和：17+5+13+21+9=65

而 n×(n^2+1)/2=5×(52+1)/2=65 可以验证，5-魔方阵中其余各行、各列及副对角线上的元素之和也都为65。

假定阵列的行列下标都从0开始，则魔方阵的生成方法为：在第0行中间置1，对从2开始的其余n2-1个数依次按下列规则存放：

(1) 假定当前数的下标为(i，j)，则下一个数的放置位置为当前位置的右上方，即下标为(i-1，j+1)的位置。

(2) 如果当前数在第0行，即i-1小于0，则将下一个数放在最后一行的下一列上，即下标为(n-1，j+1)的位置。

(3) 如果当前数在最后一列上，即j+1大于n-1，则将下一个数放在上一行的第一列上，即下标为(i-1，0)的位置。

(4) 如果当前数是n的倍数，则将下一个数直接放在当前位置的正下方，即下标为(i+1，j）的位置。

如3×3的魔方阵：

    8   1   6

    3   5   7

    4   9   2

魔方阵的排列规律如下：

(1)将1放在第一行中间一列；

(2)从2开始直到n×n止各数依次按下列规则存放；每一个数存放的行比前一个数的行数减1，列数加1（例如上面的三阶魔方阵，5在4的上一行后一列）；

(3)如果上一个数的行数为1，则下一个数的行数为n(指最下一行);例如1在第一行，则2应放在最下一行，列数同样加1；

(4)当上一个数的列数为n时，下一个数的列数应为1，行数减去1。例如2在第3行最后一列，则3应放在第二行第一列；

(5)如果按上面规则确定的位置上已有数，或上一个数是第一行第n列时，则把下一个数放在上一个数的下面。例如按上面的规定，4应该放在第1行第2列，但该位置已经被占据，所以4就放在3的下面；

#include <stdio.h>

int main()

{

    int a[100][100], n;

    scanf("%d", &n);

    int i, j, k;

    a[0][n / 2] = 1;

    // 行列

    int row, col;

    row = 0, col = n / 2;

    for (k = 2; k <= n \* n; k++)

    {

        // 上一个数是n的倍数

        if ((k - 1) % n == 0)

        {

            a[(row + 1) % n][col] = k;

            row = row + 1;

            continue;

        }

        // 当前数在第0行

        if (row == 0)

        {

            a[n - 1][(col + 1) % n] = k;

            row = n - 1;

            col = col + 1;

            continue;

        }

        // 最后一列

        if (col == n - 1)

        {

            a[row - 1][0] = k;

            row = row - 1;

            col = 0;

        }

        else

        {

            a[row - 1][col + 1] = k;

            row--;

            col++;

        }

    }

    for (i = 0; i < n; i++)

    {

        for (j = 0; j < n; j++)

        {

            printf("a[%d][%d] == %d ", i, j, a[i][j]);

        }

        printf("\n");

    }

    return 0;

}

##### 任务七:存钱

任务描述

假设银行整存整取存款不同期限的月息利率为：

0.63%  期限为1年

0.66%  期限为2年

0.69%  期限为3年

0.75%  期限为5年

0.84%    期限为8年

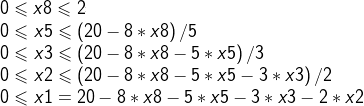
现在已知某人手上有2000元，要求通过计算选择出一种存钱方案，使得这笔钱存入银行20年后获得的利息最多，假定银行对超出存款期限的那部分时间不付利息。

任务分析

假设在这20年中，1年期限的存了x1次，2年期限的存了x2次，3年期限的存了x3 次，5年期限的存了x5次，8年期限的存了x8次，则到期时存款人所得的本利合计为（公式①）：

http://c.biancheng.net/cpp/uploads/allimg/170316/1-1F316103A0519.gif

由题意可知，显然8年期限的存款次数最多为两次，因此可得到下面对存款期限的限定条件：



#include <stdio.h>

int main()

{

    double max = 0;

    int x1, x2, x3, x5, x8;

    int cnt1, cnt2, cnt3, cnt5, cnt8;

    for (x8 = 0; x8 <= 2; x8++)

    {

        for (x5 = 0; x5 <= (20 - 8 \* x8) / 5; x5++)

        {

            for (x3 = 0; x3 <= (20 - 8 \* x8 - 5 \* x5) / 2; x3++)

            {

                for (x2 = 0; x2 <= (20 - 8 \* x8 - 5 \* x5 - 3 \* x3) / 2; x2++)

                {

                    for (int x1 = 0; x1 <= 20 - 8 \* x8 - 5 \* x5 - 3 \* x3 - 2 \* x2; x1++)

                    {

                        double res = 2000 \* pow(1 + 0.063, x1) \* pow(1 + 0.066, x2) \* pow(1 + 0.069, x3) \* pow(1 + 0.075, x5) \* pow(1 + 0.084, x8);

                        if (res > max)

                        {

                            max = res;

                            cnt1 = x1;

                            cnt2 = x2;

                            cnt3 = x3;

                            cnt5 = x5;

                            cnt8 = x8;

                        }

                    }

                }

            }

        }

    }

    printf("max == %.4f 1年期限存%d次，2年期限存%d次，3年期限存%d次，5年期限存%d次，8年期限存%d次\n", max, cnt1, cnt2, cnt3, cnt5, cnt8);

    return 0;

}

##### 任务八：三天打鱼两天晒网

问题描述

中国有句俗语叫“三天打鱼两天晒网”。某人从1990年1月1日起开始“三天打鱼两天晒网”，问这个人在以后的某一天中是“打鱼”还是“晒网”。

问题分析

根据题意可以将解题过程分为3步：

计算从1990年1月1日开始至指定日期共有多少天。

由于“打鱼”和“晒网”的周期为5天，所以将计算出的天数用5去除。

根据余数判断他是在“打鱼”还是在“晒网”，若余数为1、2、3，则他是在“打鱼”，否则是在“晒网”  
#include <stdio.h>

#include <stdbool.h>

int months[] = {0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};

bool is\_RN(int yy)

{

    if ((yy % 4 == 0 && yy % 100 != 0) || yy % 400 == 0)

    {

        return true;

    }

    return false;

}

int year\_year\_dis(int yy)

{

    int sum = 0;

    for (int i = 1990; i <= yy; i++)

    {

        if (is\_RN(i))

            sum += 366;

        else

            sum += 365;

    }

    return sum;

}

int day\_dis(int yy, int mm, int dd)

{

    if (is\_RN(yy))

    {

        months[2] = 29;

    }

    int sum = 0;

    for (int i = 1; i < mm; i++)

    {

        sum += months[i];

    }

    sum += dd;

    return sum;

}

int main()

{

    int yy, mm, dd;

    int bet1 = year\_year\_dis(yy);

    int bet2 = day\_dis(yy, mm, dd);

    int total = bet1 + bet2;

    total %= 5;

    if (total >= 1 && total <= 3)

    {

        printf("打鱼\n");

    }

    else

    {

        printf("晒网\n");

    }

    return 0;

}

##### 任务九:生兔子问题

有一对兔子，从出生后第 3 个月起每个月都生一对兔子，小兔子长到第三个月后每个月又生一对兔子，假如兔子都不死，问每个月的兔子总数为多少？

用递归实现

兔子的规律为数列：1, 1, 2, 3, 5, 8, 13, 21 ....

#include <stdio.h>

int a[100];

int f(int n)

{

    if (a[n])

        return;

    if (n == 1 || n == 2)

        a[n] = 1;

    else

        a[n] = f(n - 1) + f(n - 2);

    return a[n];

}

int main()

{

    int n, i;

    a[1] = a[2] = 1;

    scanf("%d", &n);

    int sum = f(n);

    for (i = 0; i < n; i++)

    {

        printf("第%d月兔子的兔子的总数: %d\n", i + 1, 2 \* a[i + 1]);

    }

    return 0;

}