中国高等院校计算机基础教育课程体系规划教材

C程序设计（第五版）

谭浩强 著

ISBN：978-7-302-48144-7

第 1 章

程序设计和C语言

计算机程序

指令

可以被计算机理解并执行的基本操作命令。

程序

一组计算机能识别和执行的指令。一个特定的指令序列用来完成一定的功能。

软件

与计算机系统操作有关的计算机程序、规程、规则，以及可能有的文件、文档及数据。

计算机语言

机器语言

计算机能直接识别和接受的二进制代码称为**机器指令**。机器指令的集合就是该计算机的**机器语言**。

特点：难学，难记，难检查，难修改，难以推广使用。依赖具体机器难以移植。

```
B8  7F 01
BB  21 02
03   D8
B8  1F 04
2B  C3
```

汇编语言

机器语言的符号化。用英文字母和数字表示指令的**符号语言**。

特点：相比机器语言简单好记，但仍然难以普及。

汇编指令需通过**汇编程序**转换为机器指令才能被计算机执行。依赖具体机器难以移植。

```
MOV  AX  383
MOV  BX  545
ADD   BX  AX
MOV  AX  1055
SUB    AX  BX
```

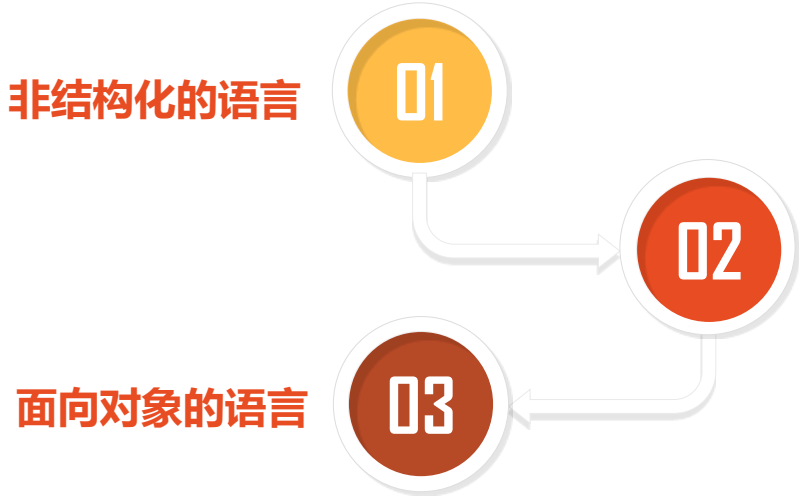
高级语言

高级语言更接近于人们习惯使用的自然语言和数学语言。

特点：功能强大，不依赖于具体机器。用高级语言编写的**源程序**需要通过**编译程序**转换为机器指令的**目标程序**。

```
S=1055-(383+545)
```

高级语言的发展



规定：

程序必须由具有良好特性的基本结构(顺序结构、选择结构、循环结构)构成，程序中的流程不允许随意跳转，程序总是由上而下顺序执行各个基本结构。

特点：

程序结构清晰，易于编写、阅读和维护。

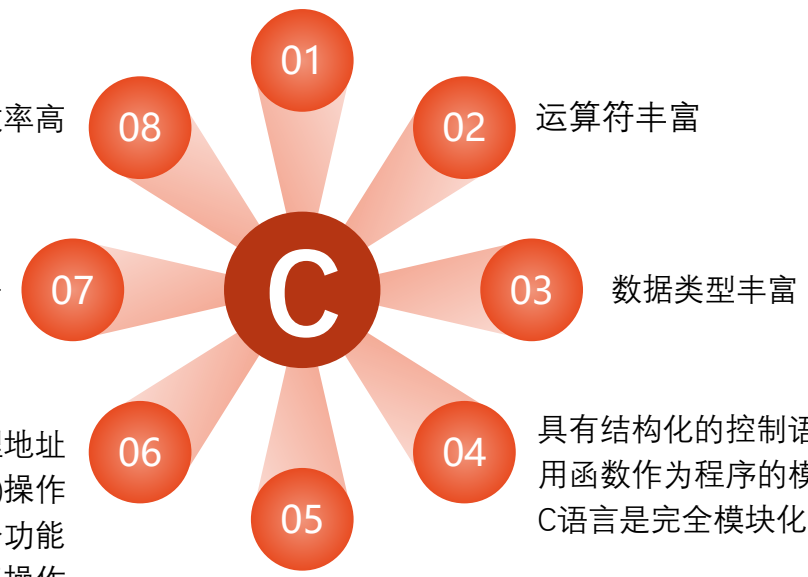
C语言的发展

D.M.Ritchie

- 1972—1973年间，美国贝尔实验室的D.M.Ritchie 在B语言的基础上设计出了C语言。
- 最初的C语言只是为描述和实现UNIX操作系统提供一种工作语言而设计的。
- 随着UNIX的日益广泛使用，C语言也迅速得到推广。1978年以后，C语言先后移植到大、中、小和微型计算机上。C语言便很快风靡全世界，成为世界上应用最广泛的程序设计高级语言。
- 以UNIX第7版中的C语言编译程序为基础，1978年，Brian W.Kernighan和Dennis M.Ritchie 合著了影响深远的名著The C Programming Language，这本书中介绍的C语言成为后来广泛使用的C语言版本的基础，它是实际上第一个C语言标准。
- 1983年，美国国家标准协会(ANSI)，根据C语言问世以来各种版本对C语言的发展和扩充，制定了第一个C语言标准草案('83 ANSI C)。
- 1989年，ANSI公布了一个完整的C语言标准——ANSI X3.159—1989(常称为ANSI C或C 89)。
- 1990年，国际标准化组织ISO(International Standard Organization)接受C 89作为国际标准ISO/IEC 9899:1990，它和ANSI的C 89基本上是相同的。
- 1999年，ISO又对C语言标准进行了修订，在基本保留原来的C语言特征的基础上，针对应用的需要，增加了一些功能，尤其是C++中的一些功能，并在2001年和2004年先后进行了两次技术修正，它被称为C 99，C 99是C 89的扩充。

目前由不同软件公司所提供的一些C语言编译系统并未完全实现C 99建议的功能，它们多以C 89为基础开发。

C语言的特点



语言简洁、紧凑，使用方便、灵活

运算符丰富

数据类型丰富

具有结构化的控制语句
用函数作为程序的模块单位，便于实现程序的模块化
C语言是完全模块化和结构化的语言

语法限制不太严格，程序设计自由度大

允许直接访问物理地址
能进行位(bit)操作
能实现汇编语言的大部分功能
可以直接对硬件进行操作
因此C语言既具有高级语言的功能，又具有低级语言的许多功能，可用来编写系统软件

程序可移植性好

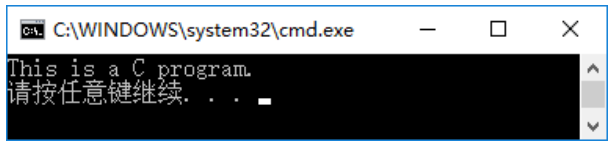
生成目标代码质量高，程序执行效率高

最简单的C语言程序

【例1.1】要求在屏幕上输出：This is a C program.

解题思路: 在主函数中用printf函数原样输出以上文字。

```
#include <stdio.h>                  //这是编译预处理指令
int main()                          //定义主函数
{                                   //函数开始的标志
    printf("This is a C program.\n");   //输出所指定的一行信息
    return 0;                       //函数执行完毕时返回函数值0
}                                   //函数结束的标志
```



最简单的C语言程序

```
#include <stdio.h>                    //这是编译预处理指令
int main()                            //定义主函数
{                                     //函数开始的标志
    printf("This is a C program.\n"); //输出所指定的一行信息
    return 0;                         //函数执行完毕时返回函数值0
}                                     //函数结束的标志
```

程序分析

- **main**是函数的名字，表示“主函数”；每一个C语言程序都必须有一个 main 函数。
- main前面的**int**表示此函数的类型是int类型(整型)，即在执行主函数后会得到一个值(即函数值)，其值为整型。
- **return 0;**的作用是当main函数执行结束前将整数0作为函数值，返回到调用函数处。
- 函数体由花括号**{}**括起来。

最简单的C语言程序

```
#include <stdio.h>                    //这是编译预处理指令
int main()                            //定义主函数
{                                     //函数开始的标志
    printf("This is a C program.\n"); //输出所指定的一行信息
    return 0;                         //函数执行完毕时返回函数值0
}                                     //函数结束的标志
```

程序分析

- **printf**是C编译系统提供的函数库中的输出函数(详见第4章)。printf函数中**双引号**内的字符串"This is a C program."按原样输出。**\n**是换行符，即在输出"This is a C program."后，显示屏上的光标位置移到下一行的开头。
- 每个语句最后都有一个**分号**，表示语句结束。

最简单的C语言程序

```
#include <stdio.h>                  //这是编译预处理指令
int main()                          //定义主函数
{                                   //函数开始的标志
    printf("This is a C program.\n");   //输出所指定的一行信息
    return 0;                       //函数执行完毕时返回函数值0
}                                   //函数结束的标志
```

程序分析

- 在使用函数库中的输入输出函数时，编译系统要求程序提供有关此函数的信息，程序第1行“**#include <stdio.h>**”的作用就是用来提供这些信息的。**stdio.h**是系统提供的一个文件名，**stdio**是standard input & output的缩写，文件后缀**.h**的意思是头文件(header file)，因为这些文件都是放在程序各文件模块的开头的。输入输出函数的相关信息已事先放在stdio.h文件中。

最简单的C语言程序

```
#include <stdio.h>                    //这是编译预处理指令
int main()                            //定义主函数
{                                     //函数开始的标志
    printf("This is a C program.\n"); //输出所指定的一行信息
    return 0;                         //函数执行完毕时返回函数值0
}                                     //函数结束的标志
```

程序分析

- //表示从此处到本行结束是“注释”，用来对程序有关部分进行必要的说明。在写C程序时应当多用注释，以方便自己和别人理解程序各部分的作用。在程序进行预编译处理时将每个注释替换为一个空格，因此在编译时注释部分不产生目标代码，注释对运行不起作用。注释只是给人看的，而不是让计算机执行的。

注释

以//开始的单行注释

这种注释可以单独占一行，也可以出现在一行中其他内容的右侧。此种注释的范围从//开始，以换行符结束。如果注释内容一行内写不下，可以用多个单行注释。

```
//第一行注释
//继续注释
```

```
/*一整块都是
注释*/
```

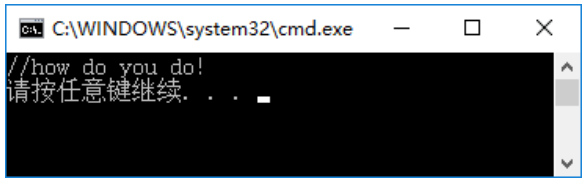
以/*开始，以*/结束的块式注释

这种注释可以包含多行内容。它可以单独占一行(在行开头以/*开始，行末以*/结束），也可以包含多行。编译系统在发现一个/*后，会开始找注释结束符*/，把二者间的内容作为注释。

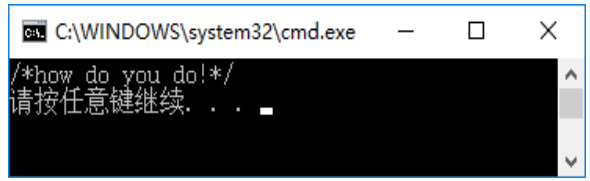
注意：在字符串中的//和/*都不作为注释的开始。而是作为字符串的一部分。

注释

```
#include <stdio.h>
int main()
{
    printf("//how do you do!\n");
    return 0;
}
```



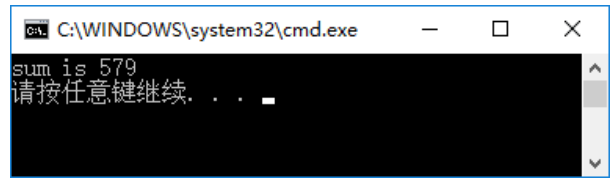

```
#include <stdio.h>
int main()
{
    printf("/*how do you do!*/\n");
    return 0;
}
```



最简单的C语言程序

【例1.2】求两个整数之和

解题思路: 设置3个变量，a和b用来存放两个整数，sum用来存放和数。用赋值运算符“=”把相加的结果传送给sum。




```
#include <stdio.h>                  //这是编译预处理指令
int main( )                         //定义主函数
{                                   //函数开始
    int a,b,sum;                    //本行是程序的声明部分，定义a,b,sum为整型变量
    a=123;                          //对变量a赋值
    b=456;                          //对变量b赋值
    sum=a+b;                        //进行a+b的运算，并把结果存放在变量sum中
    printf("sum is %d\n",sum);      //输出结果
    return 0;                       //使函数返回值为0
}                                   //函数结束
```

最简单的C语言程序

```
#include <stdio.h>                  //这是编译预处理指令
int main( )                         //定义主函数
{                                   //函数开始
   int a,b,sum;                     //本行是程序的声明部分，定义a,b,sum为整型变量
   a=123;                           //对变量a赋值
   b=456;                           //对变量b赋值
   sum=a+b;                         //进行a+b的运算，并把结果存放在变量sum中
   printf("sum is %d\n",sum);       //输出结果
   return 0;                        //使函数返回值为0
}                                   //函数结束
```

输出时用sum的值取代%d

printf("sum is %d\n", sum);

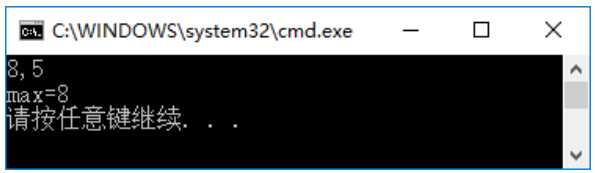
程序分析

- **printf("sum is %d\n",sum);** printf函数圆括号内有两个参数。第一个参数是双引号中的内容sum is %d\n，它是输出格式字符串，作用是输出用户希望输出的字符和输出的格式。其中sum is是用户希望输出的字符，**%d**是指定的输出格式，**d**表示用“十进制整数”形式输出。圆括号内第二个参数sum表示要输出变量sum的值。在执行printf函数时，将sum变量的值(以十进制整数表示)取代双引号中的%d。

最简单的C语言程序

【例1.3】求两个整数中的较大者

解题思路: 用一个函数来实现求两个整数中的较大者。在主函数中调用此函数并输出结果。




```
#include <stdio.h>
//主函数
int main()                          //定义主函数
{                                   //主函数体开始
    int max(int x,int y);           //对被调用函数max的声明
    int a,b,c;                      //定义变量a，b，c
    scanf("%d,%d",&a,&b);           //输入变量a和b的值
    c=max(a,b);                     //调用max函数，将得到的值赋给c
    printf("max=%d\n",c);           //输出c的值
    return 0;                       //返回函数值为0
}//主函数体结束

//求两个整数中的较大者的max函数
int max(int x,int y) //定义max函数,函数值为整型,形式参数x和y为整型
{
int z;              //max函数中的声明部分，定义本函数中用到的变量z为整型
if(x>y)z=x;         //若x>y成立，将x的值赋给变量z
else z=y;           //否则(即x>y不成立)，将y的值赋给变量z
return(z);          //将z的值作为max函数值，返回到调用max函数的位置
}
```

最简单的C语言程序

```
#include <stdio.h>
//主函数
int main()                          //定义主函数
{                                   //主函数体开始
    int max(int x,int y);           //对被调用函数max的声明
    int a,b,c;                      //定义变量a，b，c
    scanf("%d,%d",&a,&b);           //输入变量a和b的值
    c=max(a,b);                     //调用max函数，将得到的值赋给c
    printf("max=%d\n",c);           //输出c的值
    return 0;                       //返回函数值为0
}//主函数体结束

//求两个整数中的较大者的max函数
int max(int x,int y) //定义max函数,函数值为整型,形式参数x和y为整型
{
int z;                  //max函数中的声明部分，定义本函数中用到的变量z为整型
if(x>y)z=x;             //若x>y成立，将x的值赋给变量z
else z=y;               //否则(即x>y不成立)，将y的值赋给变量z
return(z);              //将z的值作为max函数值，返回到调用max函数的位置
}
```

程序分析

- 本程序包括两个函数:①主函数main；②被调用的函数max。
- max函数的作用是将x和y中较大者的值赋给变量z，最后通过return语句将z的值作为max的函数值返回给调用max函数的主函数。
- **scanf**是输入函数的名字(scanf和printf都是C的标准输入输出函数)。该scanf函数的作用是输入变量a和b的值。
- **max(a，b)**调用max函数。在调用时将a和b作为max函数的**实际参数**的值分别传送给max函数中的**形式参数**x和y。

最简单的C语言程序

```
#include <stdio.h>
//主函数
int main()                          //定义主函数
{                                   //主函数体开始
    int max(int x,int y);           //对被调用函数max的声明
    int a,b,c;                      //定义变量a，b，c
    scanf("%d,%d",&a,&b);           //输入变量a和b的值
    c=max(a,b);                     //调用max函数，将得到的值赋给c
    printf("max=%d\n",c);           //输出c的值
    return 0;                       //返回函数值为0
}//主函数体结束

//求两个整数中的较大者的max函数
int max(int x,int y) //定义max函数,函数值为整型, 形式参数x和y为整型
{
int z;              //max函数中的声明部分，定义本函数中用到的变量z为整型
if(x>y)z=x;         //若x>y成立，将x的值赋给变量z
else z=y;           //否则(即x>y不成立)，将y的值赋给变量z
return(z);          //将z的值作为max函数值，返回到调用max函数的位置
}
```

注意：本例程序中两个函数都有return语句，请注意它们的异同。

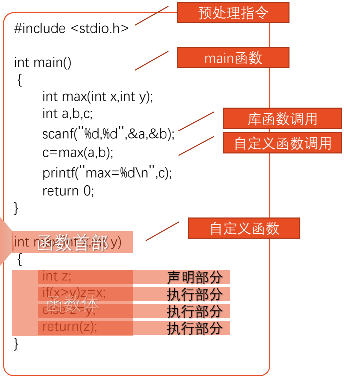
两个函数都定义为整型，都有函数值，都需要用return语句为函数指定返回值。

main函数中的return语句指定的返回值一般为0。

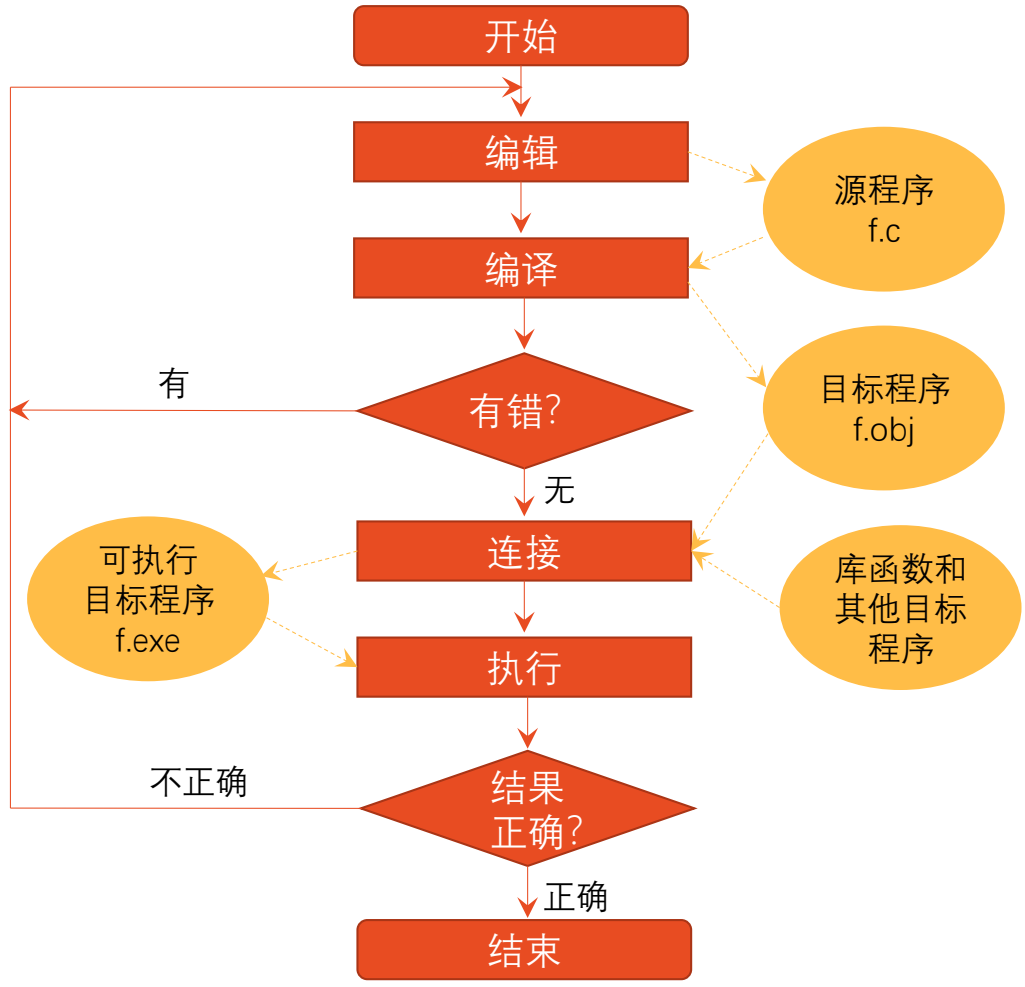
max函数的返回值是max函数中求出的两数中的最大值z，只有通过return语句才能把求出的z值作为函数的值并返回调用它的main函数中。

C语言程序的结构

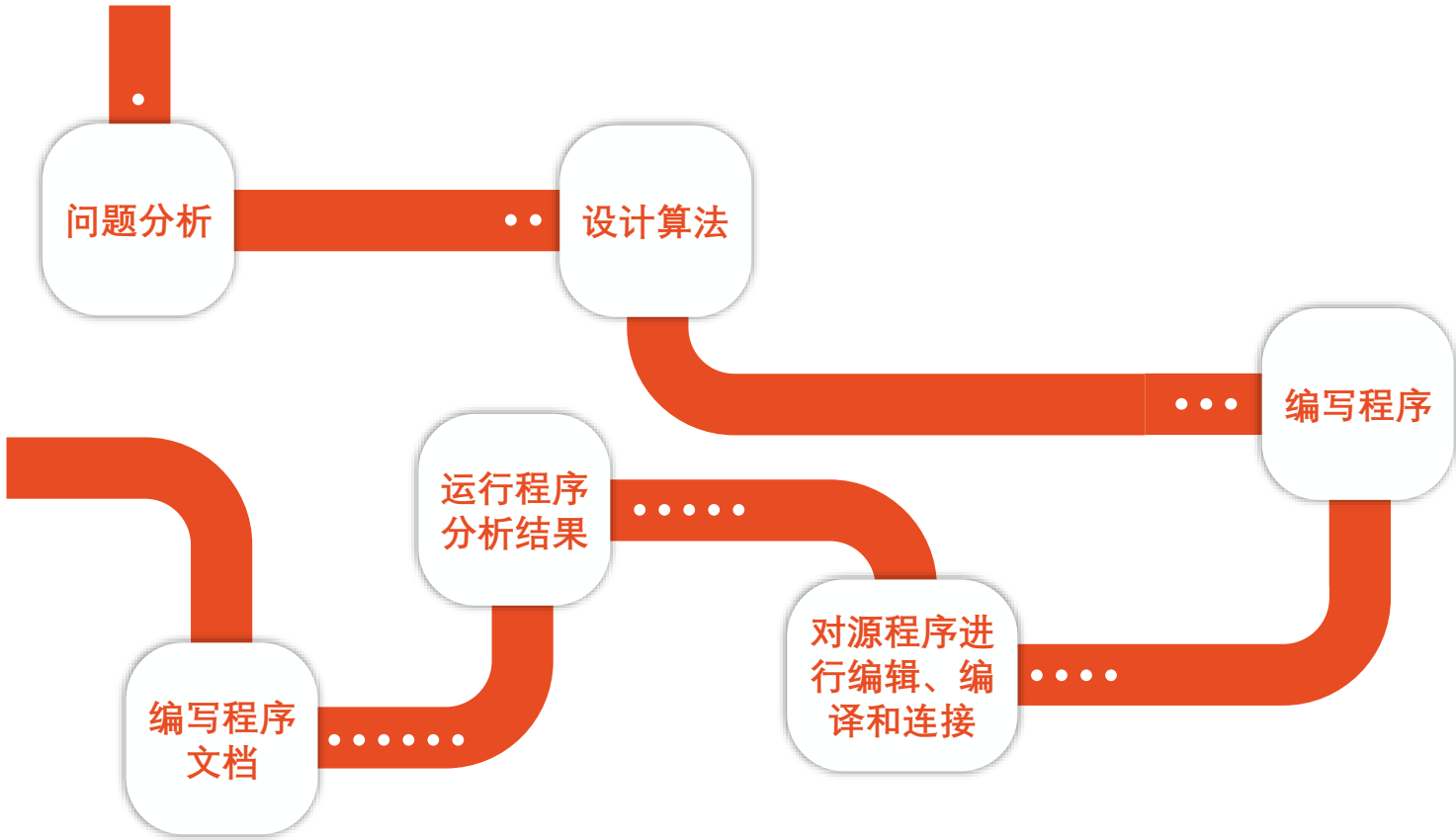
- 一个程序由一个或多个源程序文件组成
  - 源程序文件包括：预处理指令、全局声明、函数定义
- 函数时C程序的主要组成部分
  - 一个C语言程序是由一个或多个函数组成的，其中必须包含唯一一个main函数
  - 程序中被调用的函数可以是系统提供的库函数，也可以是用户根据需要自己编制设计的函数
- 一个函
  明部分
- 程序
- 程序
- 在每个数据声明和语句的最后必须有一个分号
- C语言本身不提供输入输出语句，输入输出操作由函数完成
- 程序应当包含注释



运行C程序的步骤与方法



程序设计的任务



第 2 章

算法——程序的灵魂

沃思

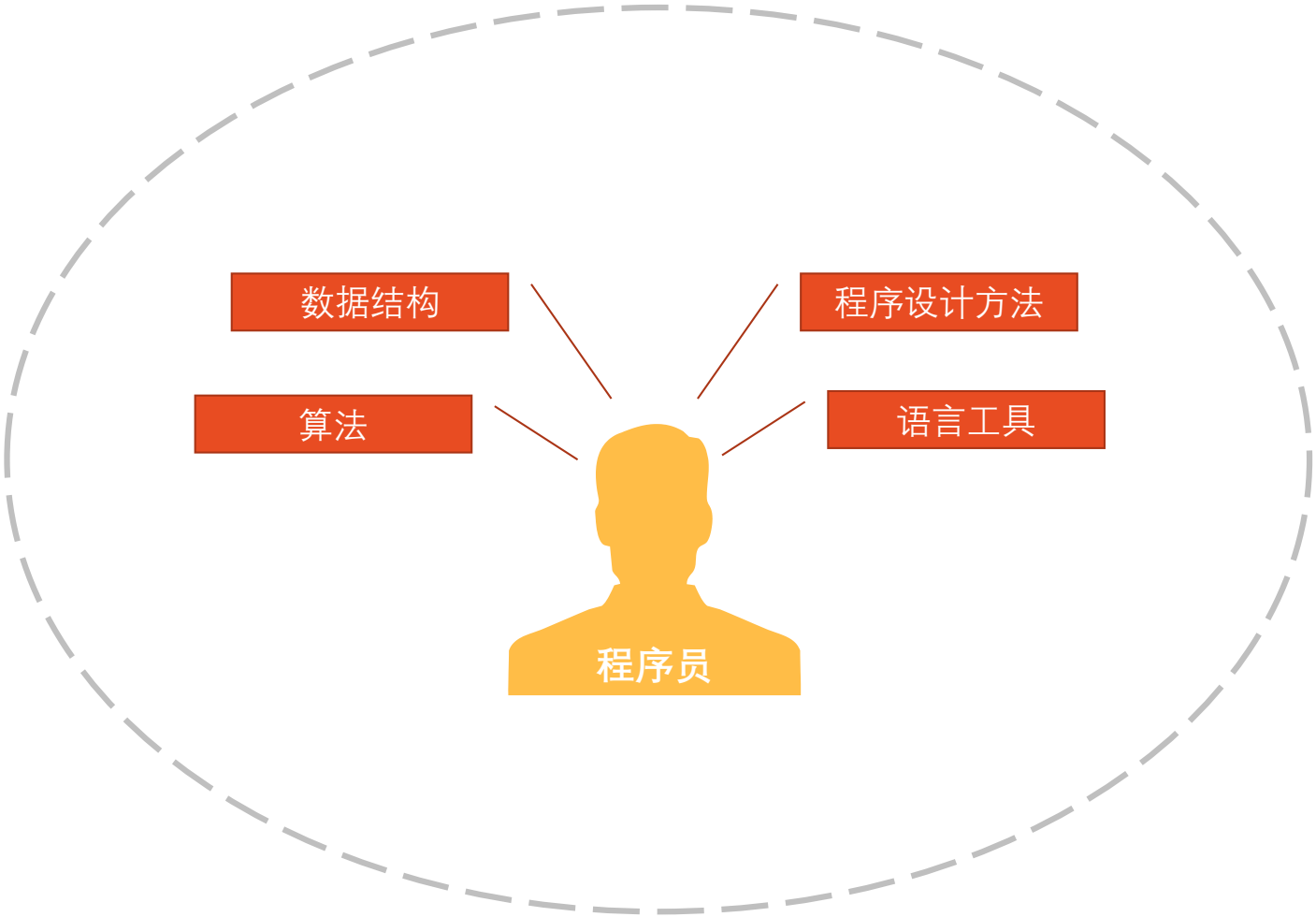
算法+数据结构=程序

数据结构

对数据的描述。在程序中要指定用到哪些数据，以及这些数据的类型和数据的组织形式。

算法

对操作的描述。即要求计算机进行操作的步骤


数据结构
程序设计方法
算法
语言工具
程序员

算 法

广义地说，为解决一个问题而采取的方法和步骤，就称为“算法”。

对同一个问题，可以有不同的解题方法和步骤。

为了有效地进行解题，不仅需要保证算法正确，还要考虑算法的质量，选择合适的算法。


算法
非数值运算算法
计算机在非数值运算方面的应用远超在数值运算方面的应用。
非数值运算的种类繁多，要求各异，需要使用者参考已有的类似算法，重新设计解决特定问题的专门算法。
数值运算算法
数值运算的目的是求数值解。
由于数值运算往往有现成的模型，可以运用数值分析方法，因此对数值运算的算法的研究比较深入，算法比较成熟。

简单的算法举例

【例2.1】求1×2×3×4×5

算法步骤

S1: 先求1乘以2，得到结果2

S2: 将步骤1得到的乘积2再乘以3，得到结果6

S3: 将6再乘以4，得24

S4: 将24再乘以5，得120

若题目改为: 求1×3×5×7×9×11

算法步骤

S1: 令p=1，或写成1=>p(表示将1存放在变量p中)

S2: 令 i=3 或写成 3=>i 表示将2存放在变量i中)

S3: 使p与i相乘，乘积仍放在变量p中，可表示为: p*i=>p

S4: 使i的值加 2 即 i+2=>i

S5: 若i≤11，返回S3；否则，结束
否则 或者 若i>11，结束；否则，返回S3

用这种方法表示的算法具有一般性、通用性和灵活性

简单的算法举例

【例2.2】有50个学生，要求输出成绩在80分以上的学生的学号和成绩

n：表示学生学号

下标i：表示第几个学生

n_1：表示第一个学生的学号

n_i：表示第i个学生的学号

g：表示学生的成绩

g_1：表示第一个学生的成绩

g_i：表示第i个学生的成绩

算法步骤

S1: 1=>i

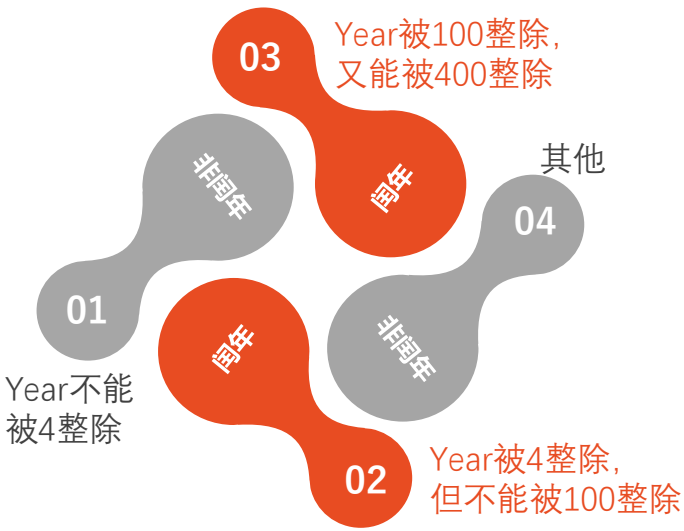
S2: 如果$g_i \geqslant 80$，则输出n_i和g_i，否则不输出

S3: i+1=>i

S4: 如果$i \leqslant 50$，返回到S2，继续执行，否则，算法结束

简单的算法举例

【例2.3】判定2000—2500年中的每一年是否为闰年，并将结果输出



算法步骤

S1: 2000=>year

S2: 若year不能被4整除，则输出year 的值和“不是闰年”。然后转到S6，检查下一个年份

S3: 若year能被4整除，不能被100整除，则输出year的值和“是闰年”。然后转到S6

S4: 若year能被400整除，输出year的值和“是闰年”，然后转到S6

S5: 输出year的值和“不是闰年”

S6: year+1=>year

S7: 当year≤2500时，转S2继续执行，否则算法停止

简单的算法举例

【例2.4】求 $1-\frac{1}{2}+\frac{1}{3}-\frac{1}{4}+\cdots+\frac{1}{99}-\frac{1}{100}$

sign：表示当前项的数值符号

term：表示当前项的值

sum：表示当前项的累加和

deno：表示当前项的分母

算法步骤

S1: sign=1

S2: sum=1

S3: deno=2

S4: sign=(-1)　sign

S5: term=sign　(1/deno)

S6: sum=sum+term

S7: deno=deno+1

S8: 若deno≤100返回S4；否则算法结束

简单的算法举例

【例2.5】给出一个大于或等于3的正整数，判断它是不是一个素数

解题思路: 所谓素数(prime)，是指除了1和该数本身之外，不能被其他任何整数整除的数。

算法步骤

S1: 输入n的值

S2: i=2（i作为除数）

S3: n被i除，得余数r

S4: 如果r=0，表示n能被i整除，则输出n“不是素数”，算法结束；否则执行S5

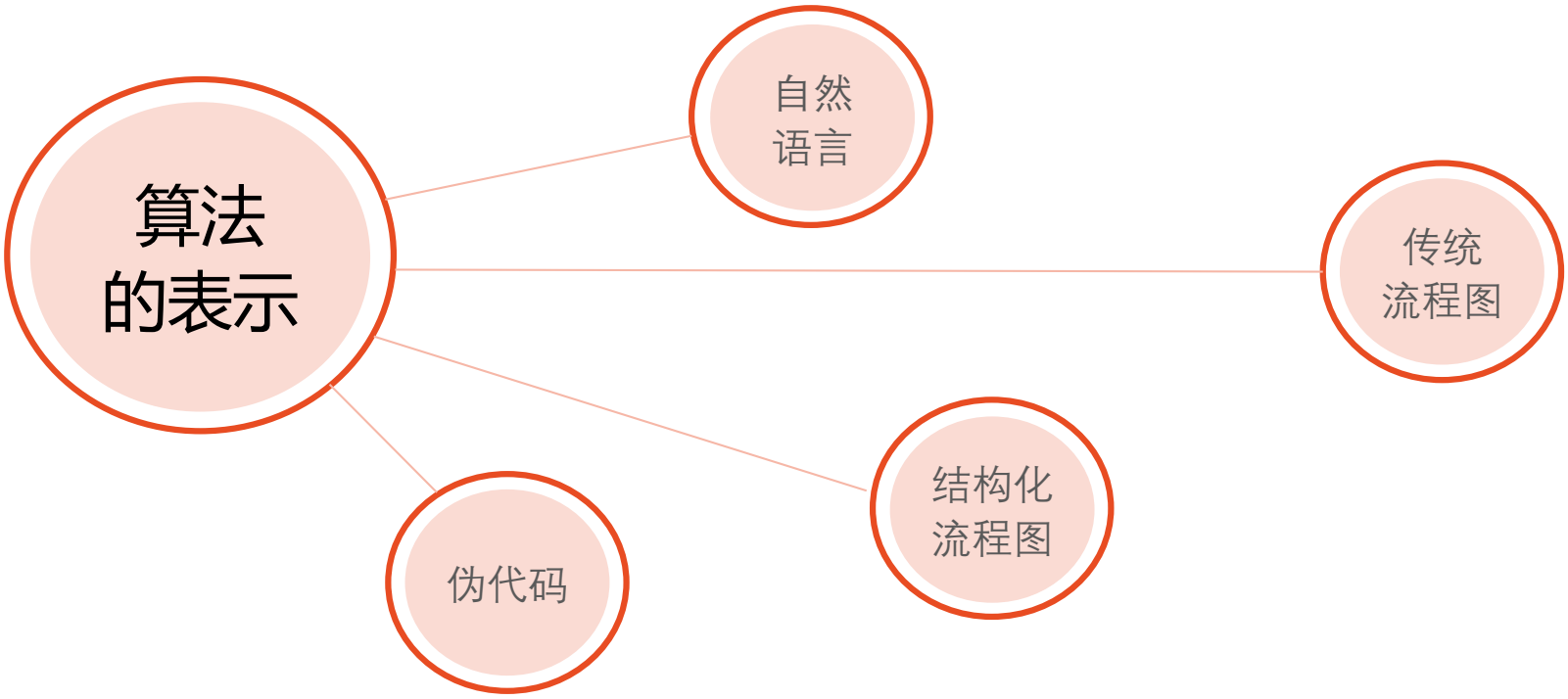
S5: i+1=>i

S6: 如果$i \leq \sqrt{n}$，返回S3；否则输出n的值以及“是素数”，然后结束

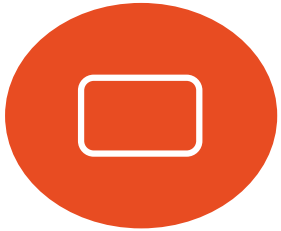
实际上，n不必被2~(n-1)之间的整数除，只须被2~n/2间整数除即可，甚至只须被2~$\sqrt{n}$之间的整数除即可。

算法的特性

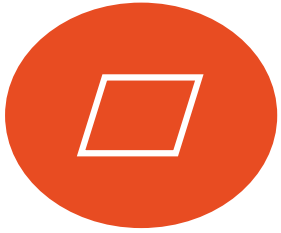
1. **有穷性**
   一个算法应包含有限的操作步骤，而不能是无限的
2. **确定性**
   算法中的每一个步骤都应当是确定的，而不应当是含糊的、模棱两可的
3. **有零个或多个输入**
   所谓输入是指在执行算法时需要从外界取得必要的信息
4. **有一个或多个输出**
   算法的目的是为了求解，“解”就是输出
5. **有效性**
   算法中的每一个步骤都应当能有效地执行，并得到确定的结果


算法
的表示
自然
语言
传统
流程图
结构化
流程图
伪代码

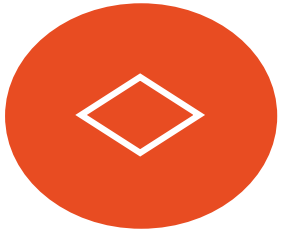
用流程图表示算法



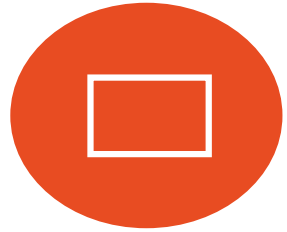
起止框



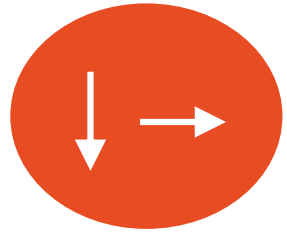
输入输出框



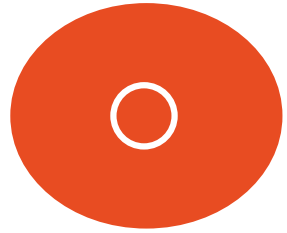
判断框



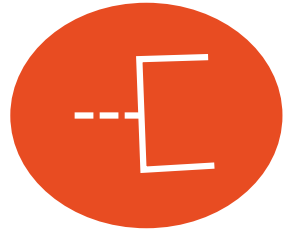
处理框



流程线



连接点



注释框

算法的流程图表示举例

【例2.6】将例2.1的算法用流程图表示。

求1×2×3×4×5。

P: 表示被乘数

i: 表示乘数

算法步骤

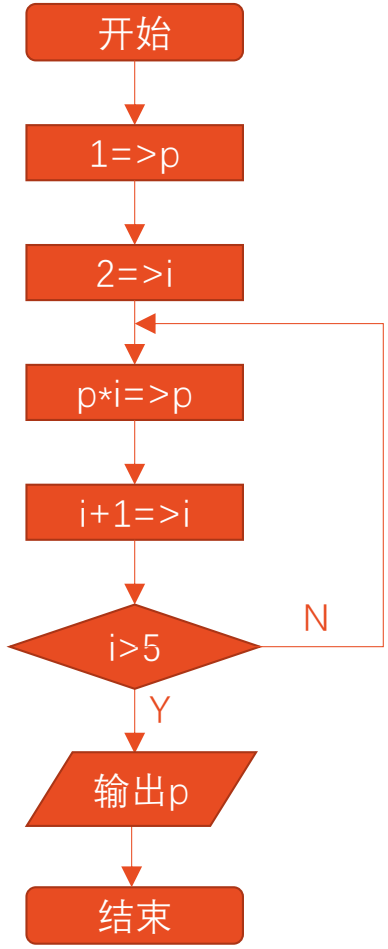
S1: 1=>p

S2: 2=>i

S3: p*i=>p

S4: i+1=>i

S5: 如果i≤5，则返回S3；否则结束



算法的流程图表示举例

【例2.7】例2.2的算法用流程图表示。

有50个学生，要求输出成绩在80分以上的学生的学号和成绩。

n：表示学生学号

下标i：表示第几个学生

n_1：表示第一个学生的学号

n_i：表示第i个学生的学号

g：表示学生的成绩

g_1：表示第一个学生的成绩

g_i：表示第i个学生的成绩

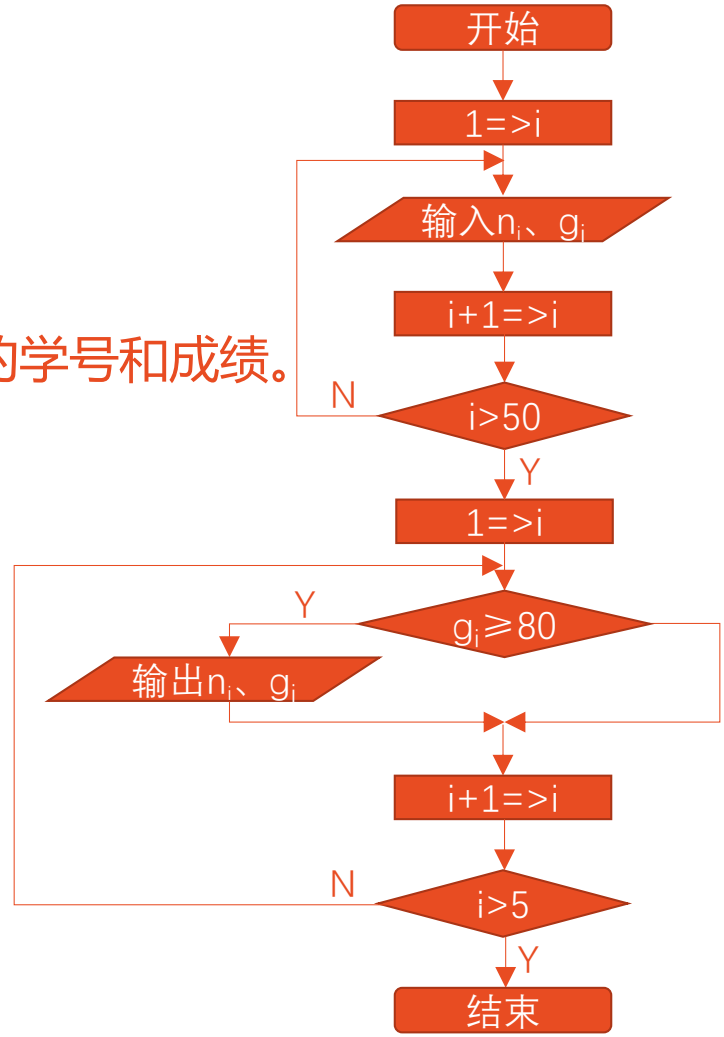
算法步骤

S1: 1=>i

S2: 如果$g_i \geqslant 80$，则输出n_i和g_i，否则不输出

S3: i+1=>i

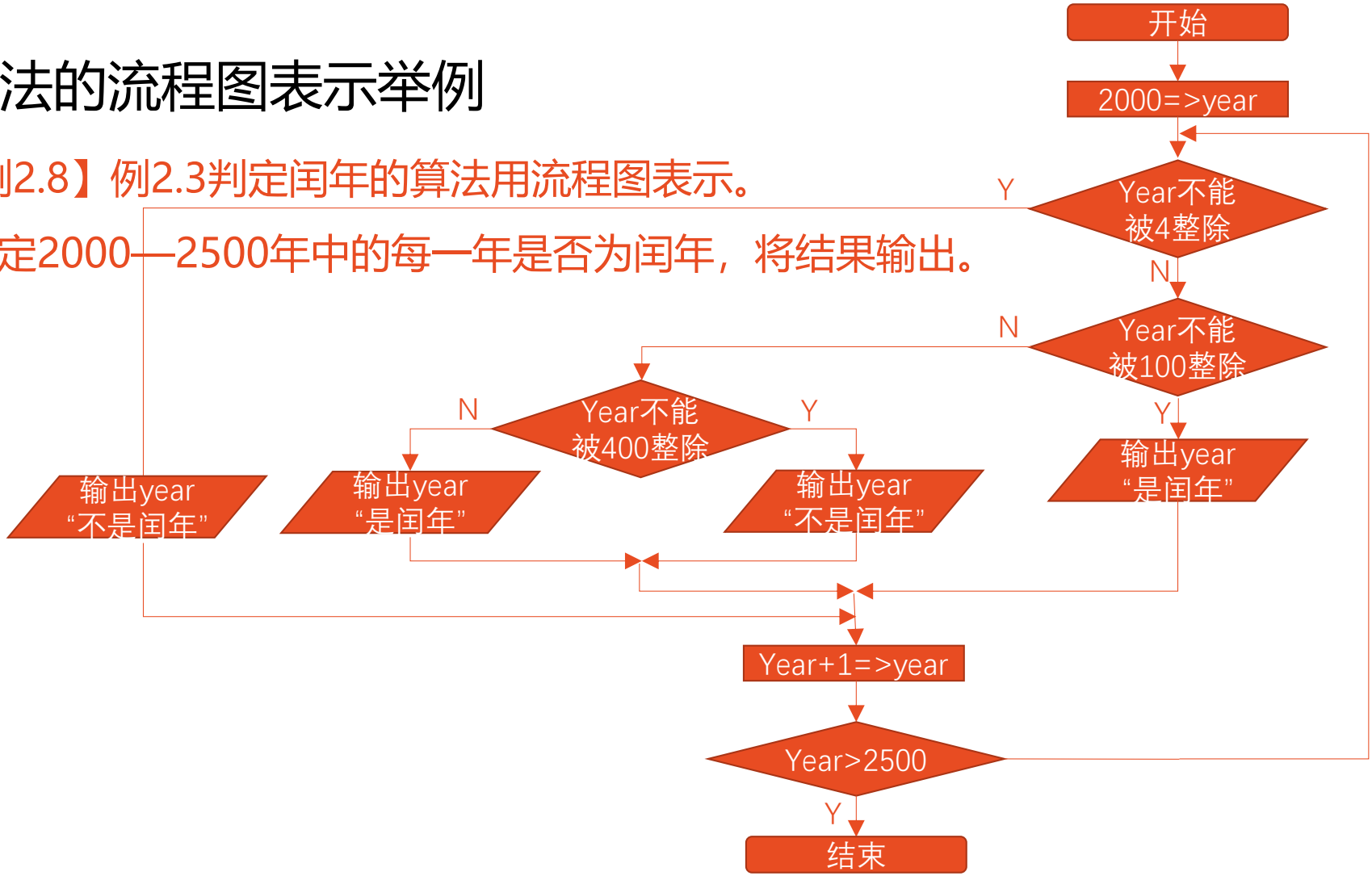
S4: 如果$i \leqslant 50$，返回到S2，继续执行，否则，算法结束



算法的流程图表示举例

【例2.8】例2.3判定闰年的算法用流程图表示。

判定2000—2500年中的每一年是否为闰年，将结果输出。



算法的流程图表示举例

【例2.9】将例2.4的算法用流程图表示。

求 $1-\frac{1}{2}+\frac{1}{3}-\frac{1}{4}+\cdots+\frac{1}{99}-\frac{1}{100}$

sign：表示当前项的数值符号

term：表示当前项的值

sum：表示当前项的累加和

deno：表示当前项的分母

算法步骤

S1: sign=1

S2: sum=1

S3: deno=2

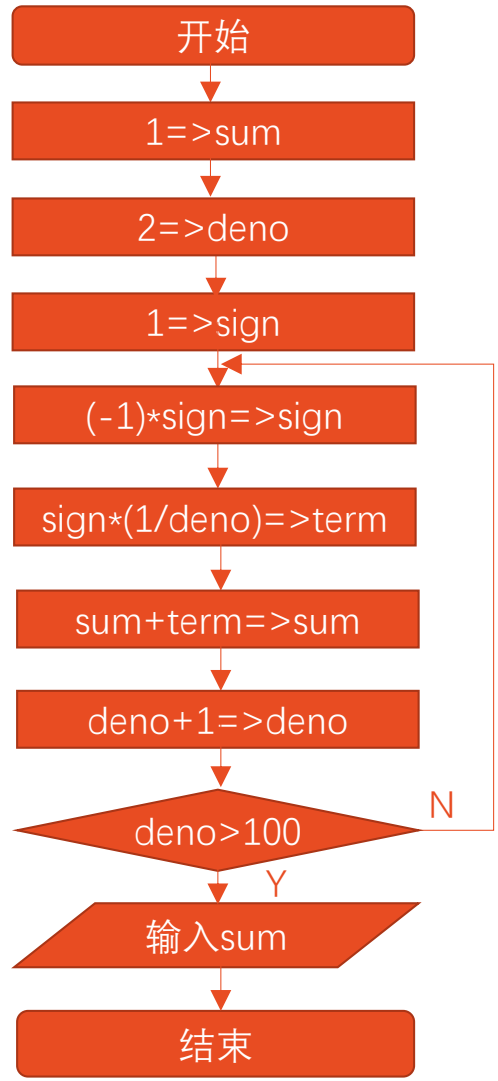
S4: sign=(-1)*sign

S5: term=sign*(1/deno)

S6: sum=sum+term

S7: deno=deno+1

S8: 若deno≤100返回S4；否则算法结束



简单的算法举例

【例2.10】例2.5判断素数的算法用流程图表示。

对一个大于或等于3的正整数，判断它是不是一个素数。

算法步骤

S1: 输入n的值

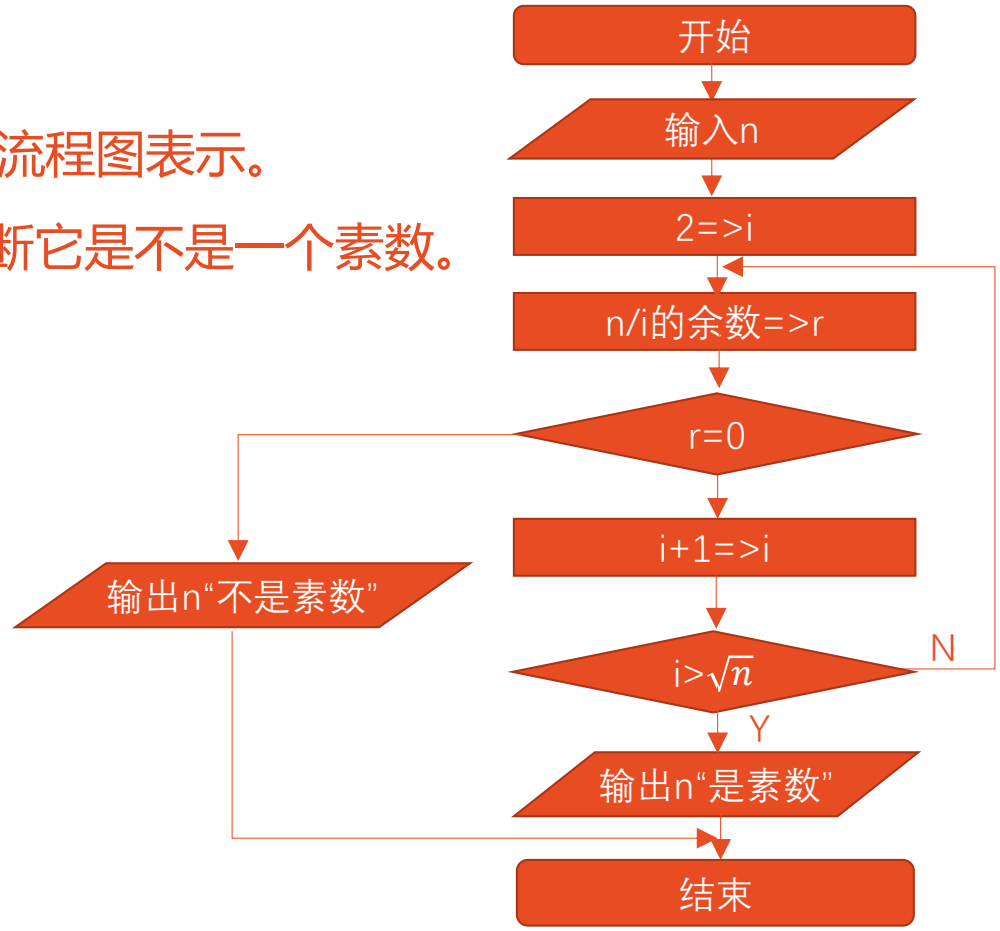
S2: i=2（i作为除数）

S3: n被i除，得余数r

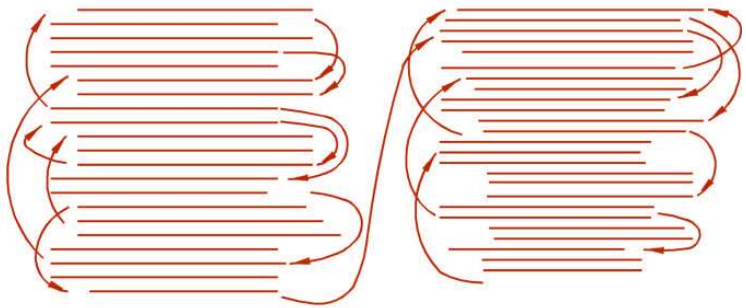
S4: 如果r=0，表示n能被i整除，则输出n“不是素数”，算法结束；否则执行S5

S5: i+1=>i

S6: 如果i≤$\sqrt{n}$，返回S3；否则输出n的值以及“是素数”，然后结束

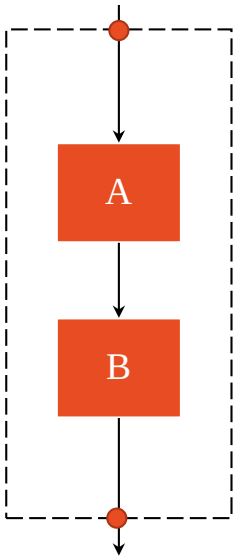


传统流程图的弊端

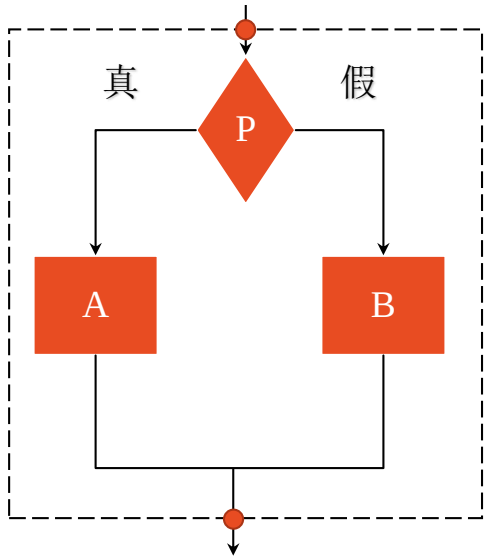


传统的流程图用流程线指出各框的执行顺序，对流程线的使用没有严格限制。因此，使用者可以不受限制地使流程随意地转来转去，使流程图变得毫无规律，阅读时要花很大精力去追踪流程，使人难以理解算法的逻辑。

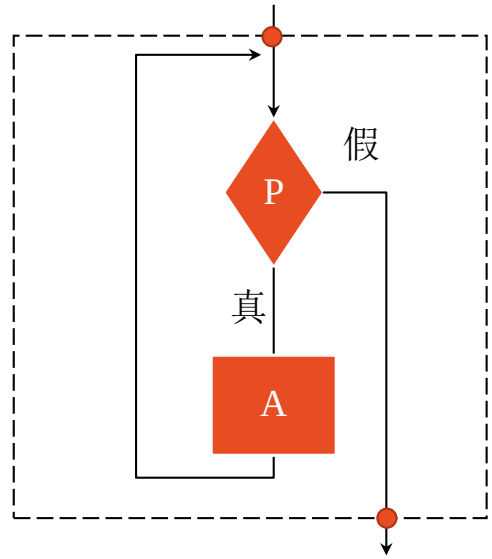
三种基本结构



顺序结构

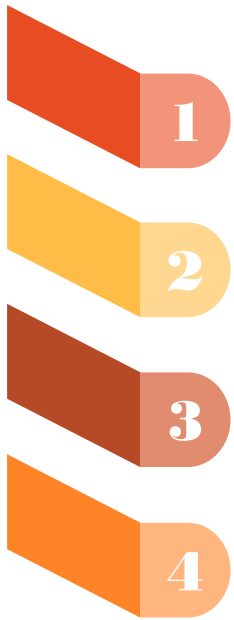


选择结构



循环结构

三种基本结构的特点



只有一个入口

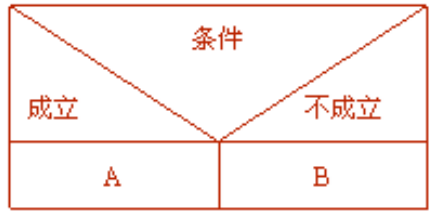
只有一个出口

结构内的每一部分都有机会被执行到

结构内不存在“死循环”

用N-S流程图表示算法

（a）顺序结构



（b）分支结构

（c）循环结构（条件在前）

（d）循环结构（条件在后）

用伪代码表示算法

伪代码是用介于自然语言和计算机语言之间的文字和符号来描述算法。它如同一篇文章一样，自上而下地写下来。每一行(或几行)表示一个基本操作。它不用图形符号，因此书写方便，格式紧凑，修改方便，容易看懂，也便于向计算机语言算法(即程序)过渡。

算法的流程图表示举例

【例2.16】求5!，用伪代码表示。

P: 表示被乘数

i: 表示乘数

算法步骤

S1: 1=>p

S2: 2=>i

S3: p*i=>p

S4: i+1=>i

S5: 如果i≤5，则返回S3；否则结束

```
begin        (算法开始)
  1=>p
  2=>I
  while i≤5
  {   p*i=>p
      i+1=>I
  }
  print p
end          (算法结束)
```

伪代码

算法的流程图表示举例

【例2.17】求 $1-\frac{1}{2}+\frac{1}{3}-\frac{1}{4}+\cdots+\frac{1}{99}-\frac{1}{100}$，用伪代码表示

sign：表示当前项的数值符号

term：表示当前项的值

sum：表示当前项的累加和

deno：表示当前项的分母

算法步骤

S1: sign=1

S2: sum=1

S3: deno=2

S4: sign=(-1)* sign

S5: term=sign*(1/deno)

S6: sum=sum+term

S7: deno=deno+1

S8: 若deno≤100返回S4；否则算法结束

```
begin       (算法开始)
   1=>sign
   1=>sum
   2=>deno
   while deno≤100
   {   (-1)*sign=>sign
       sign*(1/deno)=>term
       sum+term=>sum
       deno+1=>deno
   }
   print sum
end         (算法结束)
```

伪代码

用计算机语言表示算法

【例2.18】将例2.16表示的算法（求5!）用C语言表示。

P: 表示被乘数

i: 表示乘数

算法步骤

S1: 1=>p

S2: 2=>i

S3: p*i=>p

S4: i+1=>i

S5: 如果i≤5，则返回S3；否则结束

```
#include <stdio.h>
int main()
{
    int i,p;
    p=1;
    i=2;
    while(i<=5)
        {
            p=p*i;
            i=i+1;
        }
    printf("%d\n",p);
    return 0;
}
```

用计算机语言表示算法

【例2.18】将例2.17表示的算法求$1-\frac{1}{2}+\frac{1}{3}-\frac{1}{4}+\cdots+\frac{1}{99}-\frac{1}{100}$的值用C语言表示。

sign：表示当前项的数值符号

term：表示当前项的值

sum：表示当前项的累加和

deno：表示当前项的分母

算法步骤

S1: sign=1

S2: sum=1

S3: deno=2

S4: sign=(-1)* sign

S5: term=sign*(1/deno)

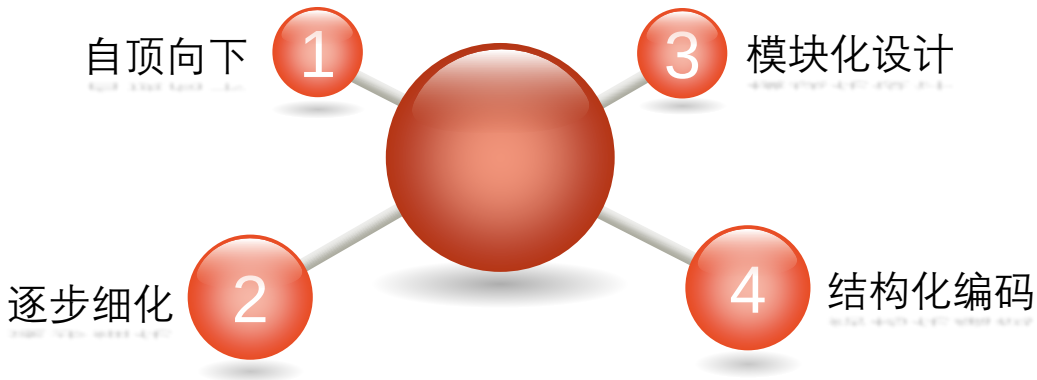
S6: sum=sum+term

S7: deno=deno+1

S8: 若deno≤100返回S4；否则算法结束

```
#include <stdio.h>
int main()
{
    int sign=1;
    double deno=2.0,sum=1.0,term;
    while(deno<=100)
        {
            sign=-sign;
            term=sign/deno;
            sum=sum+term;
            deno=deno+1;
        }
    printf("%f\n",sum);
    return 0;
}
```

结构化程序设计方法



第 3 章

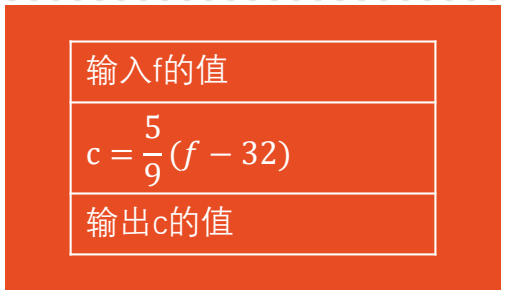
最简单的C程序设计
——顺序程序设计

顺序程序设计举例

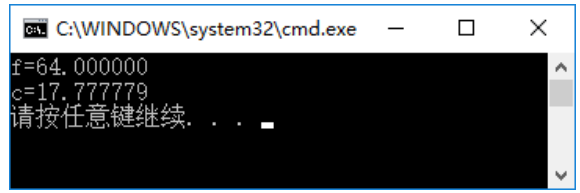
【例3.2】有人用温度计测量出用华氏法表示的温度(如64°F），今要求把它转换为以摄氏法表示的温度(如17.8℃)。

解题思路: 这个问题的算法关键在于找到二者间的转换公式。根据物理学知识，知道转换公式为：$c = \frac{5}{9}(f - 32)$，其中，f代表华氏温度，c代表摄氏温度

N-S流程图




```
#include <stdio.h>
int main()
{
    float f,c;                      //定义f和c为单精度浮点型变量
    f=64.0;                         //指定f的值
    c=(5.0/9)*(f-32);               //利用公式计算c的值
    printf("f=%f\nc=%f\n",f,c);     //输出c的值
    return 0;
}
```



顺序程序设计举例

【例3.2】计算存款利息。有1000元，想存一年。有3种方法可选:

(1)活期，年利率为r1；

(2)一年期定期，年利率为r2；

(3)存两次半年定期，年利率为r3。

请分别计算出一年后按3种方法所得到的本息和。

N-S流程图

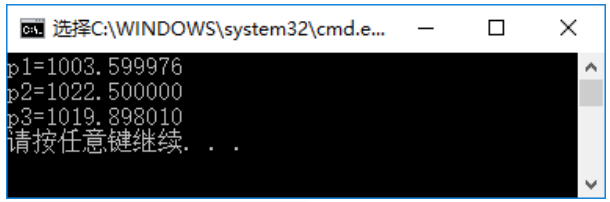
输入p0,r1,r2,r3的值
计算$p1 = p0(1 + r1)$
计算$p2 = p0(1 + r2)$
计算$p3 = p0(1 + \frac{r3}{2})(1 + \frac{r3}{2})$
输出p1,p2,p3

解题思路: 关键是确定计算本息和的公式。从数学知识可知，若存款额为p0，则:

活期存款一年后本息和为：$p1 = p0(1 + r1)$

一年期定期存款，一年后本息和为：$p2 = p0(1 + r2)$

两次半年定期存款，一年后本息和为：$p3 = p0(1 + \frac{r3}{2})(1 + \frac{r3}{2})$

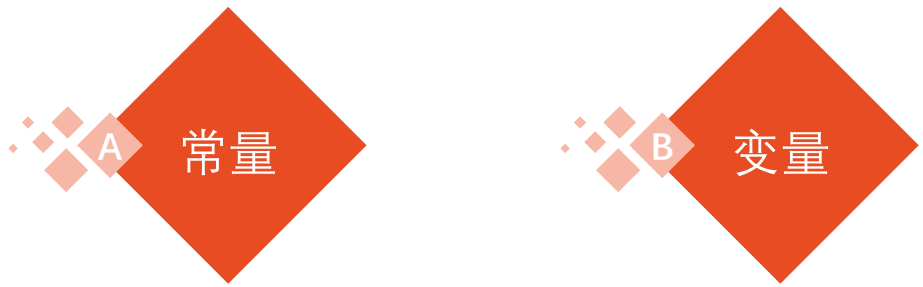



```
#include <stdio.h>
int main ()
{    float p0=1000, r1=0.0036, r2=0.0225, r3=0.0198, p1, p2, p3;
     p1=p0*(1+r1);                  //计算活期本息和
     p2=p0*(1+r2);                  //计算一年定期本息和
     p3=p0*(1+r3/2)*(1+r3/2);       //计算存两次半年定期的本息和
     printf("p1=%f\np2=%f\np3=%f\n",p1, p2, p3);    //输出结果
     return 0;
}
```

在定义实型变量的同时，对部分变量赋予初值

在输出p1，p2和p3的值之后，用\n使输出换行

在计算机高级语言中，数据的两种表现形式：



常量

1 整型常量 1000,12345,0,-345

2 实型常量 ..... **小数形式**123.456； **指数形式**12.34e3,-34.8E-23

3 字符常量 ..... **普通字符**'a','Z','#'； **转义字符**'\n','\012','\h1B'

4 字符串常量 ..... "123","boy"

5 符号常量 .....#define PI 3.1416 //注意行末没有分号

转义字符

转义序列	表示的字符		对应字符的编码值（十六进制表示）
	名称	名称缩写	
‘\a’	响铃	Bel	0x7
‘\b’	退格	BS	0x8
‘\f’	换页	FF	0xc
‘\n’	换行	LF	0xa
‘\r’	回车符	CR	0xd
‘\t’	水平制表符(横向跳格)	HT	0x9
‘\v’	垂直制表符(纵向跳格)	VT	0x6
‘\’’	单引号		0x27
‘\”’	双引号		0x22
‘\?’	问号		0x35
‘\\’	反斜线		0x5c
‘\0’	字符串结束符(空)	NULL	0x0
‘\ddd’	八进制数 ddd 表示的字符		
‘\xhhh’	十六进制数 hhh 表示的字符		

变量

变量代表一个有名字的、具有特定属性的一个存储单元。

变量用来存放数据，也就是存放变量的值。

在程序运行期间，变量的值是可以改变的。

变量必须先定义，后使用。

常变量

```
Const int a=3
```

定义a为一个整型变量，指定其值为3，而且在变量存在期间其值不能改变

常变量与常量的异同是：常变量具有变量的基本属性：有类型，占存储单元，只是不允许改变其值。可以说，常变量是有名字的不变量，而常量是没有名字的不变量。有名字就便于在程序中被引用。

```
#define Pi 3.1415926          //定义符号常量
const float pi=3.1415926;     //定义常变量
```

符号常量Pi和常变量pi都代表3.1415926，在程序中都能使用。但二者性质不同：定义符号常量用#define指令，它是预编译指令，它只是用符号常量代表一个字符串，在预编译时仅进行字符替换，在预编译后，符号常量就不存在了(全置换成3.1415926了)，对符号常量的名字是不分配存储单元的。而常变量要占用存储单元，有变量值，只是该值不改变而已。从使用的角度看，常变量具有符号常量的优点，而且使用更方便。有了常变量以后，可以不必多用符号常量。

说明：有些编译系统还未实现C 99的功能，因此不能使用常变量。

标识符

标识符就是一个对象的名字。用于标识变量、符号常量、函数、数组、类型等

标识符只能由字母、数字和下划线3种字符组成，且第1个字符必须为字母或下划线

注意

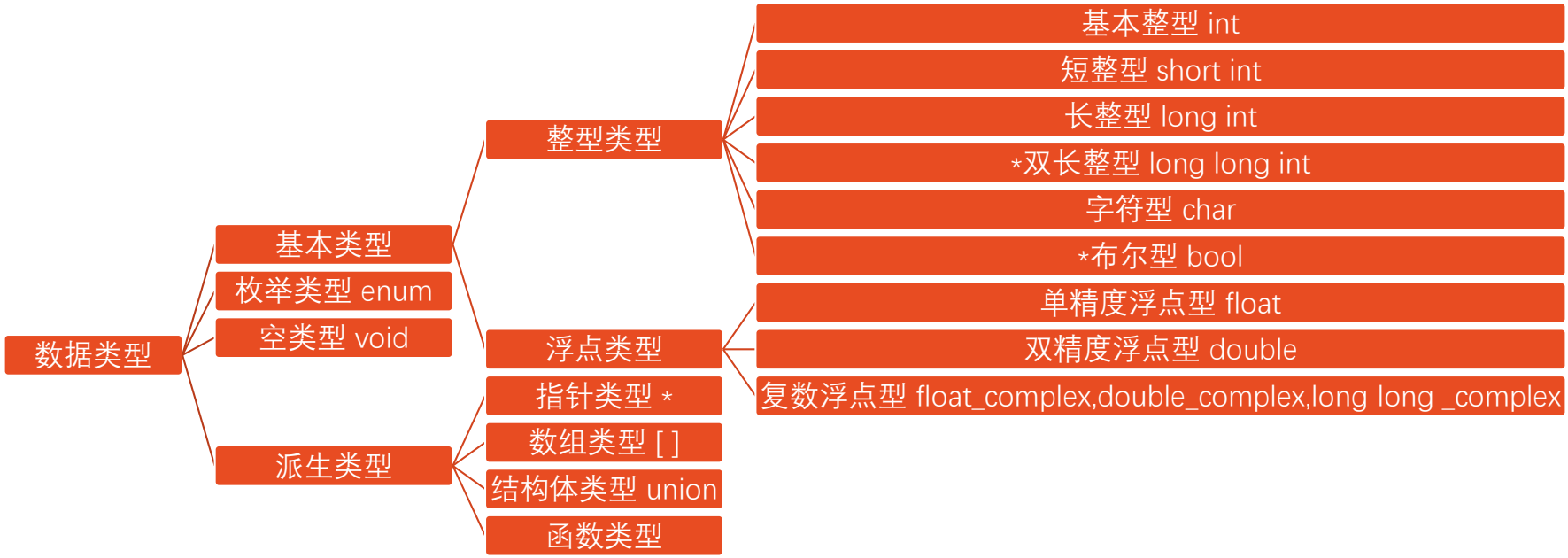
- 变量名中区分大小写字母
- 不能使用关键字作为变量名
- 变量的名字应该尽量反映变量在程序中的作用与含义

C语言中的关键字

Auto	break	case	char
const	continue	default	do
double	else	enum	extern
float	for	goto	if
int	long	register	return
short	signed	sizeof	static
struct	switch	typedef	union
unsigned	void	volatile	while

数据类型

所谓类型，就是对数据分配存储单元的安排，包括存储单元的长度(占多少字节)以及数据的存储形式。不同的类型分配不同的长度和存储形式。



计算机中带符号整型数的表示：补码

正整数的补码就是此数的二进制形式，5的补码：

0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1

负整数的补码是①将此数绝对值的二进制形式；②除最高位符号位外其他数取反；③加1。
-5的补码：

1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1

1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	0

0	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1

整型数据

整型数据类型	缺省形式的 整型数据类型	字节数	取值范围
[signed]int	int	4	-2147483648~2147483647（-2^{31}~2^{31}-1）
unsigned [int]	Unsigned	4	0~4294967295（0~2^{32}-1）
[signed] short [int]	short	2	-32768~32767（-2^{15}~2^{15}-1）
unsigned short [int]	unsigned short	2	0~65535（0~2^{16}-1）
[signed]long [int]	long	4	-2147483648~2147483647（-2^{31}~2^{31}-1）
unsigned long [int]	unsigned long	4	0~4294967295（0~2^{32}-1）
[signed]long long [int]	long long	8	-9223372036854775808~9223372036854775807（-2^{63}~2^{63}-1）
unsigned long long [int]	unsigned long long	8	0~18446744073709551615（0~2^{64}-1）

说明: C标准没有具体规定各种类型数据所占用存储单元的长度，只要求sizeof(short)≤sizeof(int)≤sizeof(long)≤sizeof(long long)，具体由各编译系统自行决定的。

sizeof是测量类型或变量长度的运算符。

整型数据

(1) 只有整型(包括字符型)数据可以加signed或unsigned修饰符，实型数据不能加。

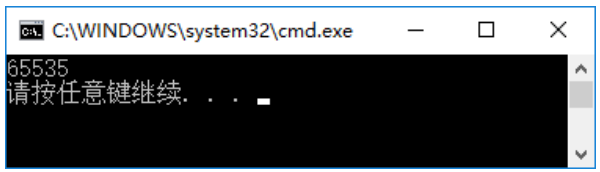
(2) 对无符号整型数据用“%u”格式输出。%u表示用无符号十进制数的格式输出。如:

```
unsigned short price=50;          //定义price为无符号短整型变量
printf("%u\n",price);             //指定用无符号十进制数的格式输出
```

在将一个变量定义为无符号整型后，不应向它赋予一个负值，否则会得到错误的结果。

如:

```
unsigned short price = -1;        //不能把一个负整数存储在无符号变量中
printf("%d\n",price);
```



字符型数据

ASCII字符集包括:

- 字母: 大写英文字母A~Z，小写英文字母a~z
- 数字: 0 ~ 9
- 专门符号: 29个,包括
- ! " # & ' ( ) * + , - . / : ; < = > ? [\ ] ^ _ ` { | } ~
- 空格符: 空格、水平制表符(tab)、垂直制表符、换行、换页(form feed)
- 不能显示的字符: 空(null)字符(以'\0'表示)、警告(以'\a'表示)、退格(以'\b'表示)、回车(以'\r'表示)等

ASCII码表

注意

字符'1'和整数1是不同的概念。

字符'1'只是代表一个形状为'1'的符号，在需要时按原样输出，在内存中以ASCII码形式存储，占1个字节。

0	0	1	1	0	0	0	1

而整数1是以整数存储方式(二进制补码方式)存储的，占2个或4个字节。

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

整数运算1+1等于整数2，而字符'1'+'1'并不等于整数2或字符'2'。

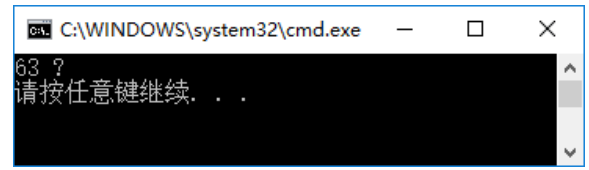
ASCII 值	控制字符	ASCII 值	控制字符	ASCII 值	控制字符	ASCII 值	控制字符
0	NUL	32	(space)	64	@	96	、
1	SOH	33	!	65	A	97	a
2	STX	34	"	66	B	98	b
3	ETX	35	#	67	C	99	c
4	EOT	36	$	68	D	100	d
5	ENQ	37	%	69	E	101	e
6	ACK	38	&	70	F	102	f
7	BEL	39	,	71	G	103	g
8	BS	40	(	72	H	104	h
9	HT	41	)	73	I	105	i
10	LF	42	*	74	J	106	j
11	VT	43	+	75	K	107	k
12	FF	44	,	76	L	108	l
13	CR	45	-	77	M	109	m
14	SO	46	.	78	N	110	n
15	SI	47	/	79	O	111	o
16	DLE	48	0	80	P	112	p
17	DCI	49	1	81	Q	113	q
18	DC2	50	2	82	R	114	r
19	DC3	51	3	83	X	115	s
20	DC4	52	4	84	T	116	t
21	NAK	53	5	85	U	117	u
22	SYN	54	6	86	V	118	v
23	TB	55	7	87	W	119	w
24	CAN	56	8	88	X	120	x
25	EM	57	9	89	Y	121	y
26	SUB	58	:	90	Z	122	z
27	ESC	59	;	91	[	123	{
28	FS	60	<	92	\	124	|
29	GS	61	=	93	]	125	}
30	RS	62	>	94	^	126	~
31	US	63	?	95	—	127	DEL

字符变量

字符变量是用类型符char定义字符变量。

```
char c='?';            //定义c为字符型变量并使初值为字符'?'。'?'的ASCII代码是63，系统把整数63赋给变量c。
```

```
printf("%d %c\n",c,c);            //用“%d”格式输出十进制整数63，用“%c”格式输出字符'?'
```

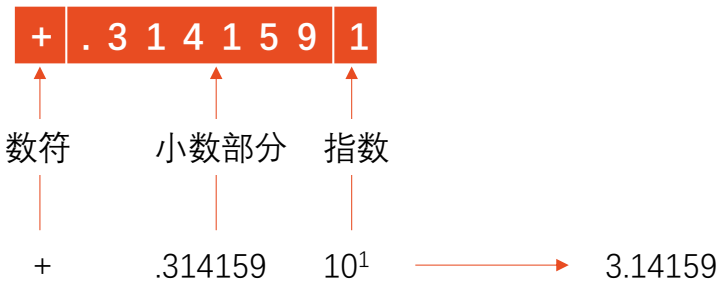


浮点型数据

$3.14159=3.14159*10^0=0.314159*10^1=314.159*10^{-2}$

由于小数点位置可以浮动，所以实数的指数形式称为**浮点数**。

浮点数类型包括float(单精度浮点型)、double(双精度浮点型)、long double(长双精度浮点型)。



注意

由于用二进制形式表示一个实数以及存储单元的长度是有限的，因此不可能得到完全精确的值，只能存储成有限的精确度。小数部分占的位（bit）数愈多，数的有效数字愈多，精度也就愈高。指数部分占的位数愈多，则能表示的数值范围愈大。

实型数据

类型	字节数	有效数字	数值范围（绝对值）
float	4	6	0以及$1.2*10^{-38}$~$3.4*10^{38}$
double	8	15	0以及$2.3*10^{-308}$~$1.7*10^{308}$
long double	8	15	0以及$2.3*10^{-308}$~$1.7*10^{308}$
	16	19	0以及$3.4*10^{-4932}$~$1.1*10^{4932}$

常量的类型

'n'——字符常量

23——整型常量

3.14159——浮点型常量

- 从常量的表示形式即可以判定其类型。
- 不带小数点的数值是整型常量，但应注意其有效范围。
- 在一个整数的末尾加大写字母L或小写字母l，表示它是长整型(long int)。
- 凡以小数形式或指数形式出现的实数均是浮点型常量，在内存中都以指数形式存储。
- C编译系统把浮点型常量都按双精度处理，分配8个字节。

常量、变量与类型

```
float a=3.14159;   //3.14159为双精度浮点常量，分配8个字节；a为float变量，分配4个字节
```

编译时系统会发出警告(warning: truncation from 'const double' to'float')，提醒用户注意这种转换可能损失精度

一般不影响结果的正确性，但会影响结果的精度。

可以在常量的末尾加专用字符，强制指定常量的类型：

```
float a=3.14159f;              //把此3.14159按单精度浮点常量处理，编译时不出现“警告”
long double a = 1.23L;         //把此1.23作为long double型处理
```

类型是变量的一个重要的属性。变量是具体存在的实体，占用存储单元,可以存放数据。而类型是变量的共性，是抽象的，不占用存储单元，不能用来存放数据。

```
int a; a=3;          //正确。对整型变量a赋值
int=3;               //错误。不能对类型赋值
```

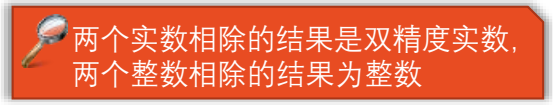
运算符和表达式

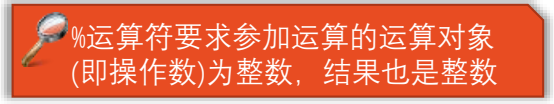
运算符

1	算术运算符	+ - * / % ++ --
2	关系运算符	> < == >= <= !=
3	逻辑运算符	! && ||
4	位运算符	<< >> ~ | ^ &
5	赋值运算符	=及其扩展赋值运算符
6	条件运算符	?:
7	逗号运算符	,
8	指针运算符	* &
9	求字节数运算符	sizeof
10	强制类型转换运算符	(类型)
11	成员运算符	. ->
12	下标运算符	[ ]
13	其他	如函数调用运算符()

常用的算数运算符

运算符	含义	举例	结果
+	正号运算符(单目运算符)	+ a	a的值
-	负号运算符(单目运算符)	-a	a的算术负值
*	乘法运算符	a*b	a和b的乘积
/	除法运算符	a / b	a除以b的商
%	求余运算符	a%b	a除以b的余数
+	加法运算符	a + b	a和b的和
–	减法运算符	a-b	a和b的差





自增（++）自减（--）运算符

++i，--i	在使用 i 之前，先使 i 的值加/减1
i++，i--	在使用 i 之后，使 i 的值加/减1

++i是先执行i=i+1，再使用i的值；而i++是先使用i的值，再执行i=i+1。

```
int i=3,j;
j=++i;    //i的值先变成4, 再赋给 j ,j的值为 4
```

```
int i=3,j;
j=i++;    //先将 i的值3赋给 j , j 的值为 3 ， 然后 i 变为 4
```

```
int i=3;
printf("%d",++i);    //输出 4
```

```
int i=3;
printf("%d",i++);    //输出3
```

建议谨慎使用++和--运算符，只用最简单的形式，即i++，i--，且把它们作为单独的表达式。

算术表达式和运算符的优先级与结合性

用算术运算符和括号将运算对象（也称操作数）连接起来的、符合C语法规则的式子称为**C算术表达式**。

运算对象包括常量、变量、函数等。

C语言规定了运算符的**优先级**(例如先乘除后加减)，还规定了运算符的**结合性**。

在表达式求值时，先按运算符的优先级别顺序执行，当在一个运算对象两侧的运算符的优先级别相同时，则按规定的“结合方向”处理。C语言规定了各种运算符的结合方向（结合性），“自左至右的结合方向”又称“**左结合性**”，即运算对象先与左面的运算符结合。相反“自右至左的结合方向”称为“**右结合性**”。

不同类型数据间的混合运算

如果一个运算符两侧的数据类型不同，则先自动进行类型转换，使二者成为同一种类型，然后进行运算。整型、实型、字符型数据间可以进行混合运算。**规律**为:

- +、-、*、/运算的两个数中有一个数为float或double型，结果是double型，因为系统将所有float型数据都先转换为double型，然后进行运算。
  - 如果int型与float或double型数据进行运算，先把int型和float型数据转换为double型，然后进行运算，结果是double型。
    - 字符(char)型数据与整型数据进行运算，就是把字符的ASCII代码与整型数据进行运算。如果字符型数据与实型数据进行运算，则将字符的ASCII代码转换为double型数据，然后进行运算。

不同类型数据间的混合运算

```
int i=3,j;
float f=2.5;
double d=7.5;
printf("%lf",10+'a'+i*f-d/3);
```

程序分析

10+'a'+i*f-d/3

① 进行10+'a'的运算，'a'的值是整数97，运算结果为107。

② 由于"*"比"+"优先级高，先进行i*f的运算。先将i与f都转成double型，运算结果为7.5，double型。

③ 整数107与 i * f 的积相加。先将整数107转换成双精度数，相加结果为114.5，double型。

④ 进行d/3的运算，先将3转换成double型，d/3结果为2.5，double型。

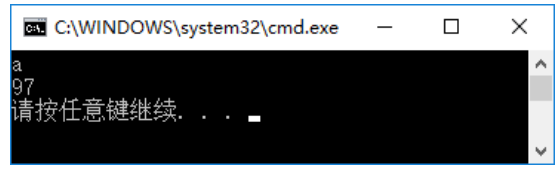
⑤ 将10+'a'+i*f的结果114.5与d/3的商2.5相减，结果为112.0，double型。

不同类型数据间的混合运算

【例3.3】给定一个大写字母，要求用小写字母输出。

解题思路: 字符数据以ASCII码存储在内存中，形式与整数的存储形式相同。 所以字符型数据和其他算术型数据之间可以互相赋值和运算。
大小写字母之间的关系是：同一个字母，用小写表示的字符的ASCII代码比用大写表示的字符的ASCII代码大32。

```
#include <stdio.h>
int main()
{
    char c1,c2;
    c1='A';              //将字符'A'的ASCII代码放到c1变量中
    c2=c1+32;            //得到字符'a'的ASCII代码，放在c2变量中
    printf("%c\n",c2);   //输出c2的值，是一个字符
    printf("%d\n",c2);   //输出c2的值，是字符'a'的ASCII代码
    return 0;
}
```




类型
转换
强制类型转换
当自动类型转换不能实现目的时，可以用强制类型转换。
自动类型转换
在运算时不必用户干预，系统自动进行的类型转换。

强制类型转换运算符

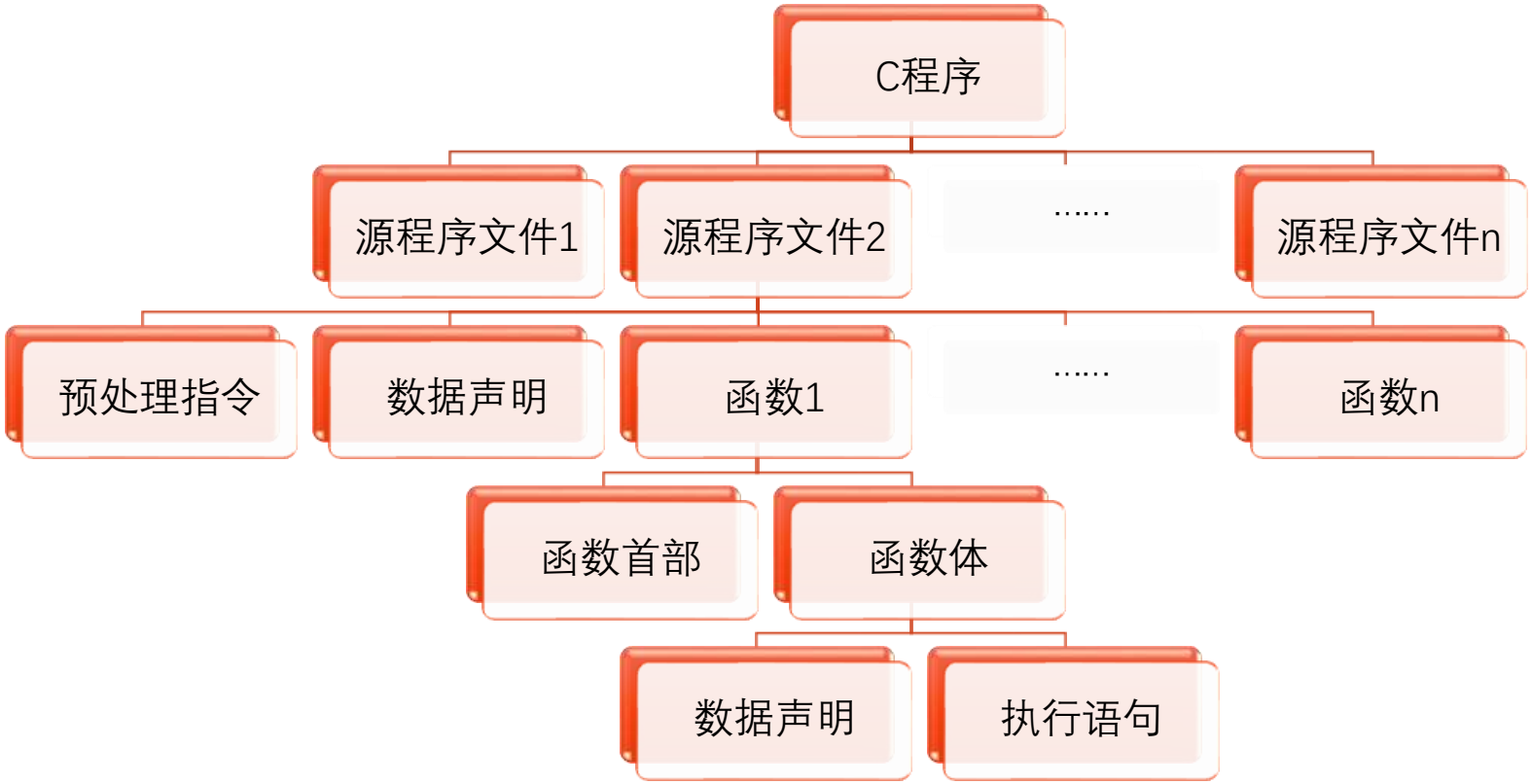
(类型名)(表达式)

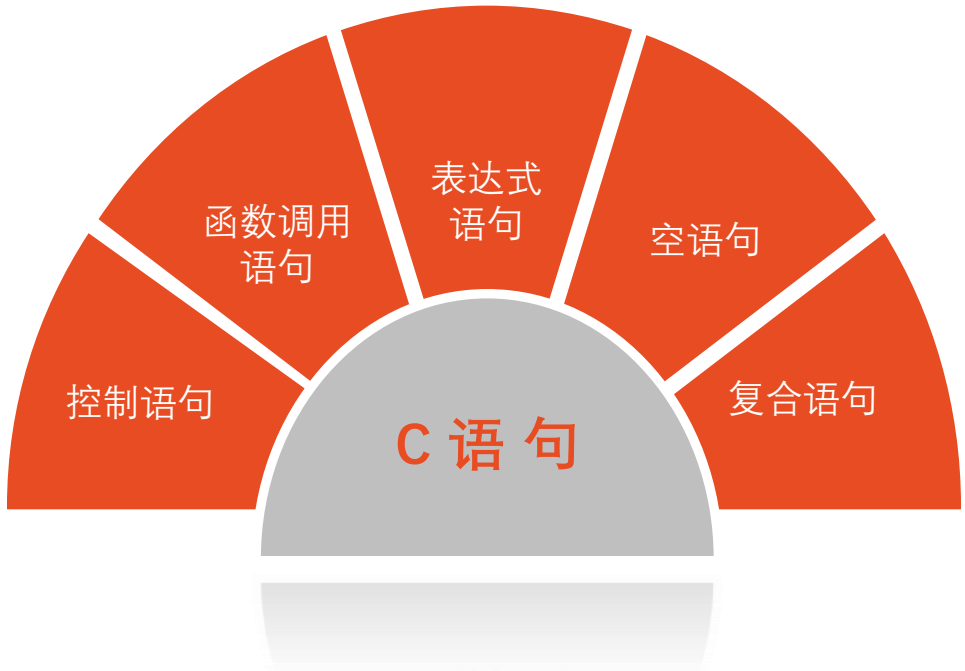
(double)a　　将 a 转换成double型
(int)(x+y)　　将x+y的值转换成int型
(float)(5%3)　　将5%3的值转换成float型
(int)x+y　　只将x转换成整型，然后与y相加

int a; float x,y;double b;
a=(int)x
进行强制类型运算(int)x后得到一个int类型的临时值，它的值等于 x 的整数部分，把它赋给a，注意x的值和类型都未变化，仍为float型。该临时值在赋值后就不再存在了。

C 语 句

C程序结构


C程序
源程序文件1
源程序文件2
……
源程序文件n
预处理指令
数据声明
函数1
……
函数n
函数首部
函数体
数据声明
执行语句


控制语句
函数调用
语句
表达式
语句
空语句
复合语句
C 语 句

① if()…else…（条件语句）

② for()…（循环语句）

③ while()…（循环语句）

④ do…while ()（循环语句）

⑤ continue（结束本次循环语句）

⑥ break（中止执行switch或循环语句）

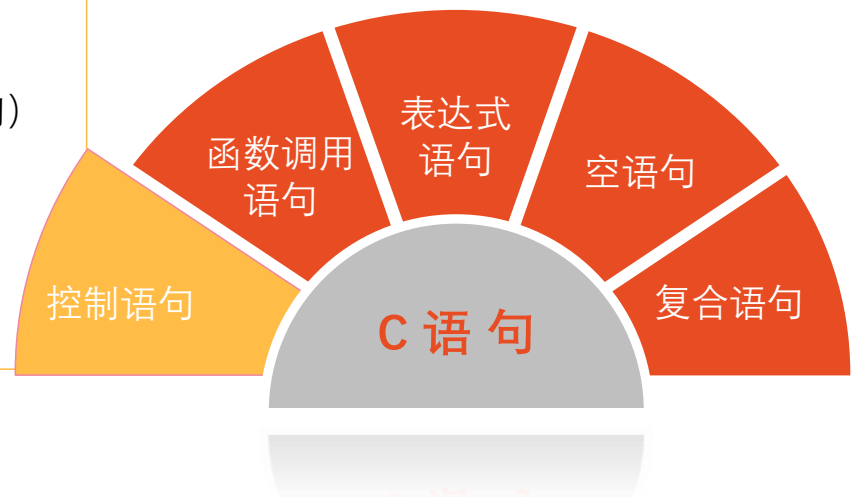
⑦ switch（多分支选择语句）

⑧ return（从函数返回语句）

⑨ goto（转向语句，在结构化程序中基本不用goto语句）

()表示括号中是一个判别条件

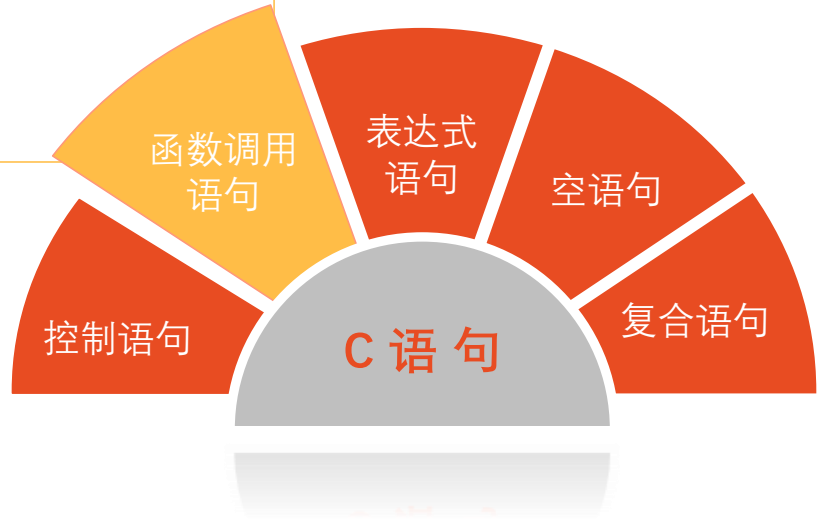
…表示内嵌的语句



函数调用语句由一个函数调用加一个分号构成。

```
printf("This is a C statement. ");
```

其中printf("This is a C statement. ")是一个函数调用，加一个分号成为一个语句。



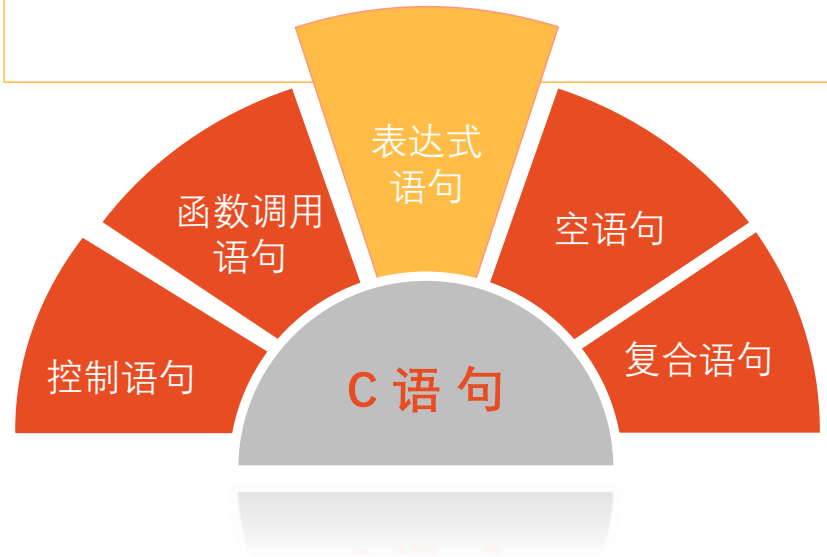
表达式语句由**一个表达式加一个分号构成**，最典型的是由赋值表达式构成一个赋值语句。例如:

a=3

是一个赋值表达式，而

a=3;

是一个赋值语句。

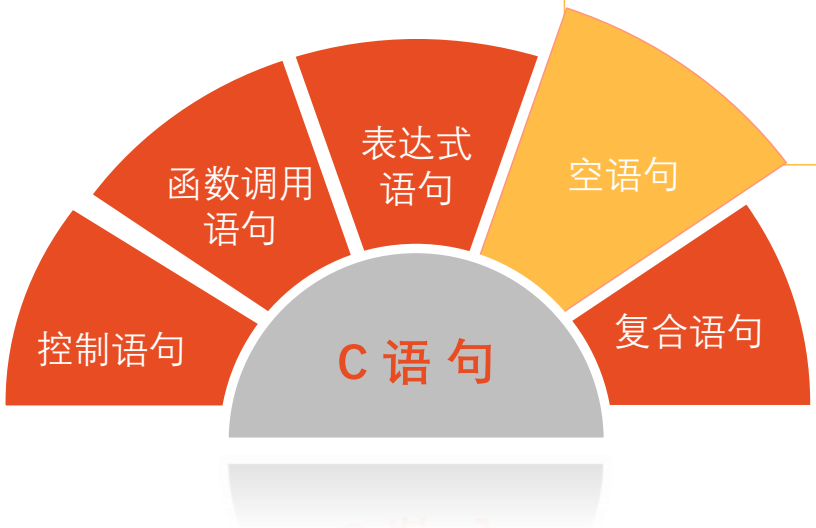


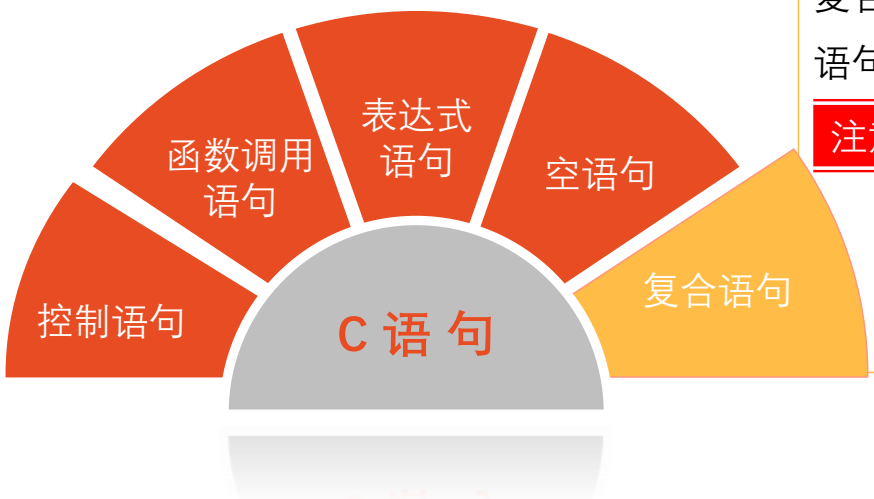
;

只有一个分号的语句即为空语句。

可以用来作为流程的转向点(流程从程序其他地方转到此语句处);

也可用来作为循环语句中的循环体（循环体是空语句，表示循环体什么也不做）。





可以用{}把一些语句和声明括起来成为复合语句(又称语句块)。

```
{
    float pi=3.14159, r=2.5, area; //定义变量
    area=pi*r*r;
    printf("area=%f",area);
}
```

复合语句常用在if语句或循环中，此时程序需要连续执行一组语句。

注意 复合语句中最后一个语句末尾的分号不能忽略不写。

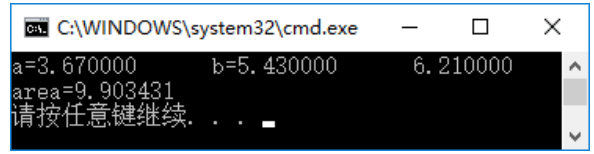
赋值语句

【例3.4】给出三角形的三边长，求三角形面积。

解题思路: 假设给定的三个边符合构成三角形的条件: 任意两边之和大于第三边。

从数学知识已知求三角形面积的公式为： area=$\sqrt{s(s-a)(s-b)(s-c)}$，其中s=(a+b+c)/2。

```
#include <stdio.h>
#include <math.h>
int main ()
{
    double a,b,c,s,area;                    //定义各变量，均为double型
    a=3.67;                                 //对边长a赋值
    b=5.43;                                 //对边长b赋值
    c=6.21;                                 //对边长c赋值
    s=(a+b+c)/2;                            //计算s
    area=sqrt(s*(s-a)*(s-b)*(s-c));         //计算area
    printf("a=%f\tb=%f\t%f\n",a,b,c);       //输出三边a,b,c的值
    printf("area=%f\n",area);               //输出面积area的值
    return 0;
}
```



为提高精度，变量都定义为双精度类型

sqrt函数是求平方根的函数。由于要调用数学函数库中的函数，必须在程序的开头加一条#include指令，把头文件“math.h”包含到程序中来。

转移字符'\t'用来调整输出的位置，使输出的数据清晰、整齐、美观

赋值运算符“=”

“=”的作用是将一个数据赋给一个变量。

例如：a=3的作用是执行一次赋值操作（或称赋值运算）。把常量3赋给变量a。

也可以将一个表达式的值赋给一个变量。

*复合赋值运算符

在赋值符=之前加上其他运算符，可以构成复合的运算符。

a+=3	等价于a=a+3
x*=y+8	等价于x=x*(y+8)
x%=3	等价于x=x%3

凡是二元（二目）运算符，都可以与赋值符一起组合成复合赋值符。

有关算术运算的复合赋值运算符有+=， -=， =， /=， %=。

注意

如果赋值符右边是包含若干项的表达式，则相当于它**有括号**。例如，x%=y+3等价于x=x%(y+3)，切勿错写为x=x%y+3。

赋值表达式： 变量 赋值运算符 表达式

赋值表达式的作用是将一个表达式的值赋给一个变量，因此赋值表达式具有计算和赋值的双重功能。

对赋值表达式求解的**过程**是:

①求赋值运算符右侧的“表达式”的值，②赋给赋值运算符左侧的变量。既然是一个表达式，就应该有一个值，**表达式的值等于赋值后左侧变量的值**。

赋值运算符左侧应该是一个可修改值的“**左值**”(left value，简写为lvalue)。

能出现在赋值运算符右侧的表达式称为“**右值**”(right value，简写为rvalue)。

注意

并不是任何形式的数据都可以作为左值的，**左值应当为存储空间并可以被赋值**。变量可以作为左值，而算术表达式a+b就不能作为左值，常量也不能作为左值。

赋值表达式： 变量 赋值运算符 表达式

a=(b=5)

括号内的b=5是一个赋值表达式，它的值等于5。执行表达式“a=(b=5)”，就是执行b=5和a=b两个赋值表达式。因此a的值等于5，整个赋值表达式的值也等于5。赋值运算符按照“自右而左”的结合顺序，因此，(b=5)外面的括号可以不要，即a=(b=5)和a=b=5等价，都是先求b=5的值（得5），然后再赋给a。

a=b=c=5	表达式值为5，a,b,c值均为5
a=5+(c=6)	表达式值为11，a值为11，c值为6
a=(b=4)+(c=6)	表达式值为10，a值为10，b等于4，c等于6
a=(b=10)/(c=2)	表达式值为5，a等于5，b等于10，c等于2
a=(b=3*4)	表达式值为12，a,b值均为12

赋值表达式使得赋值操作不仅可以出现在赋值语句中，而且可以出现在其他语句中(如输出语句、循环语句等)

如: printf("%d", a=b);

如果b的值为3，则输出a的值(也是表达式a=b的值)为3。在一个printf函数中完成了赋值和输出双重功能。

*赋值过程中的类型转换

如果赋值运算符两侧的类型一致，则直接进行赋值。

```
int i;
i=234;    //直接将整数234存入变量i的存储单元中
```

如果赋值运算符两侧的类型不一致，但都是基本类型时，在赋值时要进行类型转换。类型转换是由系统自动进行的，转换的规则是：

1. 将浮点型数据（包括单、双精度）赋给整型变量时，先对浮点数取整，即舍弃小数部分，然后赋予整型变量。
2. 将整型数据赋给单、双精度变量时，数值不变，但以浮点数形式存储到变量中。
3. 将一个double型数据赋给float变量时，先将双精度数转换为单精度，即只取6～7位有效数字，存储到float型变量的4个字节中。应注意双精度数值的大小不能超出float型变量的数值范围；将一个float型数据赋给double型变量时，数值不变，在内存中以8个字节存储，有效位数扩展到15位。
4. 字符型数据赋给整型变量时，将字符的ASCII代码赋给整型变量。
5. 将一个占字节多的整型数据赋给一个占字节少的整型变量或字符变量时，只将其低字节原封不动地送到被赋值的变量（即发生“截断”）。

赋值表达式和赋值语句

C语言的赋值语句属于表达式语句，由一个赋值表达式加一个分号组成。

在一个表达式中可以包含另一个表达式。

```
if ((a=b)>0)max=a;
/*先进行赋值运算（将b的值赋给a），然后判断a是否大于0，如大于0，执行max=a。
请注意，在if语句中的a=b不是赋值语句，而是赋值表达式。*/
```

注意

区分赋值表达式和赋值语句。

赋值表达式的末尾没有分号，而赋值语句的末尾必须有分号。在一个表达式中可以包含一个或多个赋值表达式，但绝不能包含赋值语句。

变量赋初值

可以用赋值语句对变量赋值，也可以在定义变量时对变量赋以初值。

```
int a=3;          //指定a为整型变量，初值为3；相当于int a; a=3;
float f=3.56;     //指定f为浮点型变量，初值为3.56
char c='a';       //指定c为字符变量，初值为'a'
int a,b,c=5;      //指定a, b ,c为整型变量，但只对c初始化，c的初值为 5 ；
                  //相当于int a,b,c; c=5;
```

对几个变量赋予同一个初值：

```
int a=3,b=3,c=3;
```

```
int a=b=c=3;        //可以先定义，再用赋值语句，即int a,b,c; a=b=c=3;
```

数据的输入输出

输入输出举例

【例3.5】求$ax^2+bx+c=0$方程的根。a,b,c由键盘输入，设$b^2-4ac>0$。

解题思路: 首先要知道求方程式的根的方法。由数学知识已知: 如果$b^2-4ac\geq 0$，则一元二次方程有两个实根：$x1=\frac{-b+\sqrt{b^2-4ac}}{2a}$，$x2=\frac{-b-\sqrt{b^2-4ac}}{2a}$，将分式分为两项：$p=\frac{-b}{2a}$，$q=\frac{\sqrt{b^2-4ac}}{2a}$，则x1=p+q，x2=p-q，有了这些式子，只要知道a,b,c的值，就能顺利地求出方程的两个根。

```
#include <stdio.h>
#include<math.h>                    //程序中要调用求平方根函数sqrt
int main()
{    double a,b,c,disc,x1,x2,p,q;   //disc用来存放判别式(b    b-4ac)的值
     scanf("%lf%lf%lf",&a,&b,&c);
     disc=b*b-4*a*c;
     p=-b/(2.0*a);
     q=sqrt(disc)/(2.0*a);
     x1=p+q;x2=p-q;
     printf("x1=%7.2f\nx2=%7.2f\n",x1,x2);
     return 0;
}
```

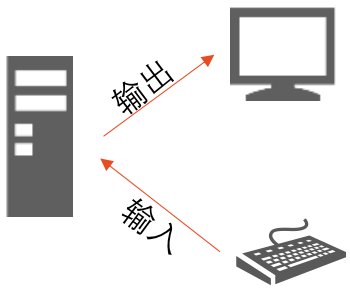
```
C:\WINDOWS\system32\cmd.exe
1 3 2
x1=  -1.00
x2=  -2.00
请按任意键继续. . .
```

scanf函数用于输入a,b,c的值。
函数中括号内变量a,b,c的前面，要用地址符**&**。&a表示变量a在内存中的地址。
双引号内用**%lf**格式声明，表示输入的是**双精度**型实数。
格式声明为"%lf%lf%lf"，要求输入3个双精度实数。程序运

在printf函数中，在格式符f的前面加了"**7.2**"，表示在输出x1和x2时，指定数据占7列，其中小数占2列。优点：
①可以根据实际需要来输出小数的位数;
②如果输出多个数据，可使输出数据整齐美观。

有关输入输出的概念

1 输入输出是以计算机主机为主体而言的



2 C语言本身不提供输入输出语句

输入和输出操作是由C标准函数库中的函数来实现的。

优点:

简化编译系统简化

增强通用性和可移植性

3 要在程序文件的开头用预处理指令#include把有关头文件放在本程序中

#include<stdio.h>

#include指令说明

三种形式:

#include "c:\cpp\include\myfile.h" ❶
#include "myfile.h"❷ ❸
#include <myfile.h> ❸

❶按指定路径查找文件
❷源程序文件所在目录
❸C编译系统指定的include目录

printf函数

用来向终端（或系统隐含指定的输出设备）输出若干个任意类型的数据。

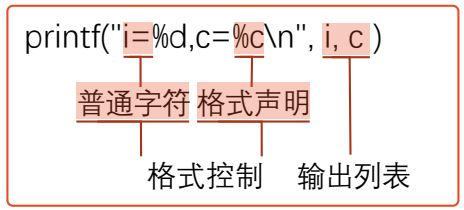
printf（格式控制，输出表列）

(1) “**格式控制**”是用双引号括起来的一个字符串，称为格式控制字符串，简称格式字符串。包括:

① **格式声明**。格式声明由“%”和格式字符组成。作用是将输出的数据转换为指定的格式后输出。

② **普通字符**。普通字符即需要在输出时原样输出的字符。

(2) **输出表列**是程序需要输出的一些数据，可以是常量、变量或表达式。



printf函数——格式声明

% 附加字符 格式字符

(1) printf函数输出时，务必注意输出对象的类型应与上述格式说明匹配，否则将会出现错误。

(2) 除了X,E,G外，其他格式字符必须用小写字母，如%d不能写成%D。

(3) 可以在printf函数中的格式控制字符串内包含转义字符，如\n,\t,\b,\r,\f和\377等。

(4) 一个格式声明以“%”开头，以格式字符之一为结束，中间可以插入附加格式字符（也称修饰符）。

(5) 如果想输出字符“%”，应该在“格式控制字符串”中用连续两个“%”表示，如：printf(”%f%%\n”,1.0/3);

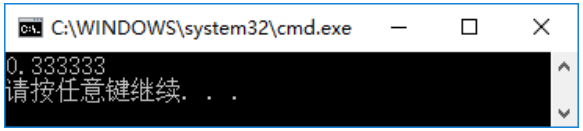
格式字符	说　明
d,i	以带符号的十进制形式输出整数（正数不输出符号）
o	以八进制无符号形式输出整数（不输出前导符０）
x,X	以十六进制无符号形式输出整数（不输出前导符0x），用x则输出十六进制数的a～f时以小写形式输出，用X时，则以大写字母输出
u	以无符号十进制形式输出整数
c	以字符形式输出，只输出一个字符
s	输出字符串
f	以小数形式输出单、双精度数，隐含输出6位小数
e,E	以指数形式输出实数，用e时指数以“e”表示(如1.2e+02)，用E时指数以“E”表示(如1.2E+02)
g,G	选用%f或%e格式中输出宽度较短的一种格式，不输出无意义的0。用G时，若以指数形式输出，则指数以大写表示

附加字符	说　明
l	长整型整数，可加在格式符ｄ、ｏ、ｘ、ｕ前面）
m (代表一个正整数)	数据最小宽度
n (代表一个正整数)	对实数，表示输出ｎ位小数；对字符串，表示截取的字符个数
-	输出的数字或字符在域内向左靠

printf函数举例

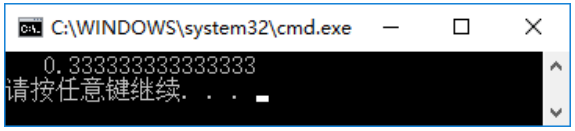
【例3.6】用%f输出实数，只能得到 6 位小数。

```
#include <stdio.h>
int main()
{    double a=1.0;
     printf("%f\n",a/3);
     return 0;
}
```



虽然a是双精度型，a/3的结果也是双精度型，但是用%f格式声明只能输出6位小数。

```
#include <stdio.h>
int main()
{    double a=1.0;
     printf("%20.15f\n",a/3);
     return 0;
}
```



一个双精度数只能保证15位有效数字的精确度，即使指定小数位数为50(如用%55.50f)，也不能保证输出的50位都是有效的数字。

注意

在用%f输出时要注意数据本身能提供的有效数字，如float型数据的存储单元只能保证6位有效数字。double型数据能保证15位有效数字。

scanf函数

用来输入数据。

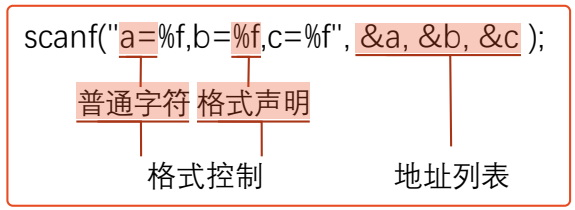
scanf（格式控制，地址表列）

(1) “**格式控制**”是用双引号括起来的一个字符串，含义同printf函数。包括：

① **格式声明**。以%开始，以一个格式字符结束，中间可以插入附加的字符。

② **普通字符**。

(2) **地址表列**是由若干个地址组成的表列，可以是变量的地址，或字符串的首地址。



scanf函数——格式声明

% 附加字符 格式字符

(1) scanf函数中的格式控制后面应当是**变量地址**，而不是变量名。

应与上述格式说明匹配，否则将会出现错误。

(2)如果在格式控制字符串中除了格式声明以外还有其他字符，则在输入数据时在对应的位置上应输入与这些字符相同的字符。

(3)在用“%c”格式声明输入字符时，空格字符和“转义字符”中的字符都作为有效字符输入。

(4) 在输入数值数据时，如输入空格、回车、Tab键或遇非法字符(不属于数值的字符)，认为该数据结束。

格式字符	说　明
d,i	输入有符号的十进制整数
u	输入无符号的十进制整数
o	输入无符号的八进制整数
x,X	输入无符号的十六进制整数(大小写作用相同)
c	输入单个字符
s	输入字符串，将字符串送到一个字符数组中，在输入时以非空白字符开始，以第一个空白字符结束。字符串以串结束标志'\0'作为其最后一个字符
f	输入实数，可以用小数形式或指数形式输入
e,E,g,G	与f作用相同，e与f、g可以互相替换(大小写作用相同)

附加字符	说　明
l	输入长整型数据（可用%ld，%lo，%lx，%lu）以及double型数据（用%lf或%le）
h	输入短整型数据（可用%hd，%ho，%hx）
域宽	指定输入数据所占宽度（列数），域宽应为正整数
*	本输入项在读入后不赋给相应的变量


getchar
输入
字符
函数
输出
putchar

putchar函数

从计算机向显示器输出一个字符。

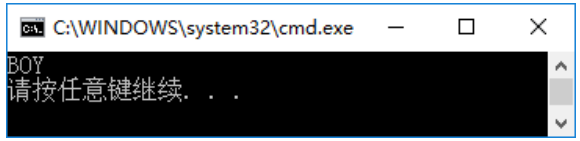
putchar(c)

用putchar函数既可以输出**可显示字符**，也可以输出**控制字符**和**转义字符**。

putchar(c)中的c可以是**字符常量、整型常量、字符变量**或**整型变量**(其值在字符的ASCII代码范围内)。

【例3.8】先后输出BOY三个字符。

解题思路: 定义3个字符变量，分别赋以初值'B'，'O'，'Y'，然后用putchar函数输出这3个字符变量的值。




```
#include <stdio.h>
int main()
{
    char a='B',b='O',c='Y';    //定义3个字符变量并初始化
    putchar(a);                //向显示器输出字符B
    putchar(b);                //向显示器输出字符O
    putchar(c);                //向显示器输出字符Y
    putchar ('\n');            //向显示器输出一个换行符
    return 0;
}
```

```
#include <stdio.h>
int main()
{
    int a=66,b=79,c=89;
    putchar(a);
    putchar(b);
    putchar(c);
    putchar ('\n');
    return 0;
}
```

getchar函数

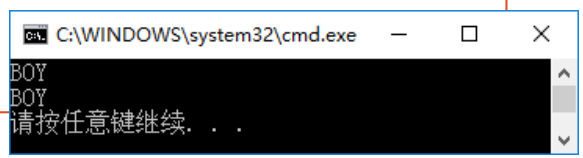
向计算机输入一个字符。

getchar()

【例3.9】从键盘输入BOY 3个字符，然后把它们输出到屏幕。

解题思路: 用3个getchar函数先后从键盘向计算机输入BOY 3个字符，然后用putchar函数输出。

```
#include <stdio.h>
int main()
{    char a,b,c;   //定义字符变量a,b,c
     a=getchar();//从键盘输入一个字符，送给字符变量a
     b=getchar();//从键盘输入一个字符，送给字符变量b
     c=getchar();//从键盘输入一个字符，送给字符变量c
     putchar(a);  //将变量a的值输出
     putchar(b);  //将变量b的值输出
     putchar(c);  //将变量c的值输出
     putchar('\n');//换行
     return 0;
}
```



函数**没有参数**。

函数的值就是从输入设备得到的字符。

只能接收一个字符。

如果想输入多个字符就要用多个函数。

不仅可以从输入设备获得一个可显示的字符，而且可以获得控制字符。

用getchar函数得到的字符可以赋给一个字符变量或整型变量，也可以作为表达式的一部分。如，putchar(getchar());将接收到的字符输出。

getchar函数

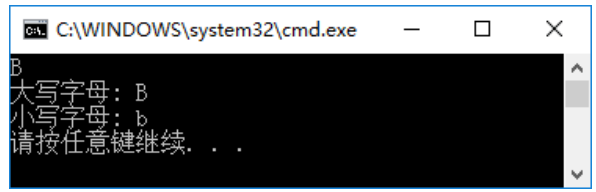
【例3.10】改写例3.3程序，使之可以适用于任何大写字母。

从键盘输入一个大写字母，在显示屏上显示对应的小写字母。

解题思路: 用getchar函数从键盘读入一个大写字母，把它转换为小写字母，然后用putchar函数输出该小写字母。

用printf函数输出

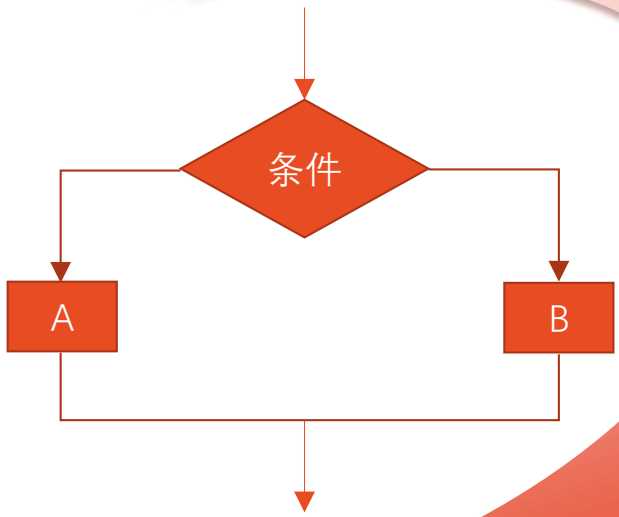
```
#include <stdio.h>
int main ()
{
    char c1,c2;
    c1=getchar();    //从键盘读入一个大写字母，赋给字符变量c1
    c2=c1+32;  //得到对应的小写字母的ASCII代码，放在字符变量c2中
    printf("大写字母: %c\n小写字母: %c\n",c1,c2);  //输出c1,c2的值
    return 0;
}
```



第 4 章

选择结构程序设计

选择结构和条件判断



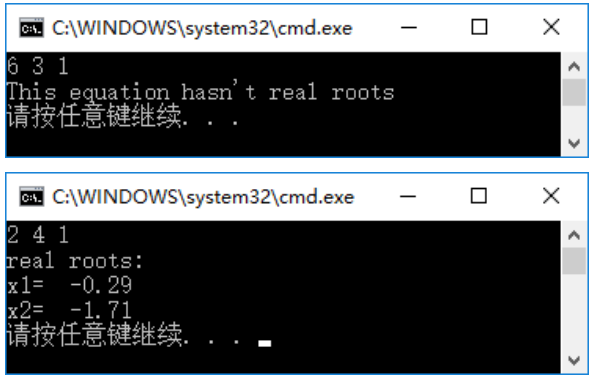
C语言有两种选择语句

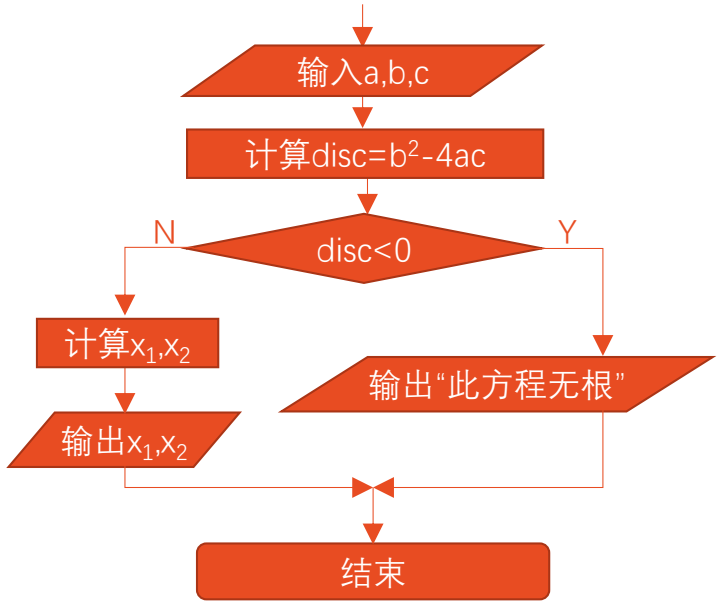
- **if语句**，用来实现**两个分支**的选择结构
- **switch**语句，用来实现**多分支**的选择结构

if语句例题

【例4.1】在例3.5的基础上对程序进行改进。题目要求解得$ax^2+bx+c=0$方程的根。由键盘输入a,b,c。假设a,b,c的值任意，并不保证$b^2-4ac\geq 0$。需要在程序中进行判别，如果$b^2-4ac\geq 0$，就计算并输出方程的两个实根，如果$b^2-4ac<0$，就输出“此方程无实根”的信息。

```
#include<stdio.h>
#include<math.h>       //程序中要调用求平方根函数sqrt
int main()
{     double a,b,c,disc,x1,x2,p,q;   //disc是判别式sqrt(b*b-4ac)
      scanf("%lf%lf%lf",&a,&b,&c);   //输入双精度浮点型变量的值要用格式声明"%lf"
      disc=b*b-4*a*c;
      if(disc<0)                     //若b*b-4ac<0
            printf("This equation hasn't real roots\n");  //输出“此方程无实根”
      else                           //b*b-4ac≥0
      {     p=-b/(2.0*a);
            q=sqrt(disc)/(2.0*a);
            x1=p+q;x2=p-q;           //求出方程的两个根
            printf("real roots:\nx1=%7.2f\nx2=%7.2f\n",x1,x2);     //输出方程的两个根
      }
      return 0;
}
```



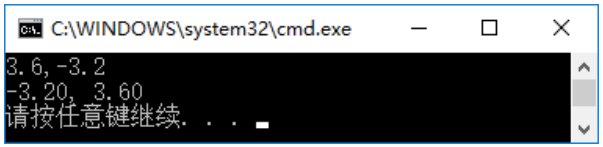


用if语句实现选择结构

【例4.2】输入两个实数，按由小到大的顺序输出这两个数。

解题思路: 只要做一次比较，然后进行一次交换即可。用if语句实现条件判断。

```
#include <stdio.h>
int main()
{
    float a,b,t;
    scanf("%f,%f",&a,&b);
    if(a>b)
    {                   //将a和b的值互换
        t=a;
        a=b;
        b=t;
    }
    printf("%5.2f,%5.2f\n",a,b);
    return 0;
}
```



两个变量值的互换

```
a=b; //把变量b的值赋给变量a，a的值等于b的值
b=a; //再把变量a的值赋给变量b，变量b值没有改变
```

因此，为了实现互换，必须借助于第三个变量

用if语句实现选择结构

【例4.3】输入3个数a，b，c，要求按由小到大的顺序输出。

```
#include <stdio.h>
int main()
{
    float a,b,c,t;
    scanf("%f,%f,%f",&a,&b,&c);
    if(a>b)
    {   t=a;      //借助变量t，实现变量a和变量b互换值
        a=b;
        b=t;
    }             //互换后，a小于或等于b
    if(a>c)
    {   t=a;      //借助变量t，实现变量a和变量c互换值
        a=c;
        c=t;
    }             //互换后，a小于或等于c
    if(b>c)       //还要
    {   t=b;      //借助变量t，实现变量b和变量c互换值
        b=c;
        c=t;
    }             //互换后，b小于或等于c
    printf("%5.2f,%5.2f,%5.2f\n",a,b,c);      //顺序输出a,b,c的值
    return 0;
}
```

算法步骤

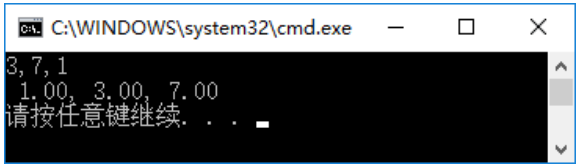
S1: if a>b，将a和b对换
(交换后，a是a、b中的小者)

S2: if a>c，将a和c对换
(交换后，a是a、c中的小者，因此a是三者中最小者)

S3: if b>c，将b和c对换
(交换后，b是b、c中的小者，也是三者中次小者)

S4: 顺序输出a，b，c

在经过第1次互换值后，a≤b，经过第2次互换值后a≤c，这样a已是三者中最小的(或最小者之一)，但是b和c谁大还未解决，还需要进行比较和互换。经过第3次互换值后，a≤b≤c。



if语句的一般形式

if (表达式) 语句1
[else 语句2]

“表达式”可以是关系表达式、逻辑表达式，甚至是数值表达式

方括号内的部分(即else子句)为可选的，既可以有，也可以没有

语句1和语句2可以是一个简单的语句，也可以是一个复合语句，还可以是另一个if语句

形式1　没有else子句部分

```
if(表达式) 语句1
```

形式2　有else子句部分

```
if (表达式)
        语句1
else
        语句2
```

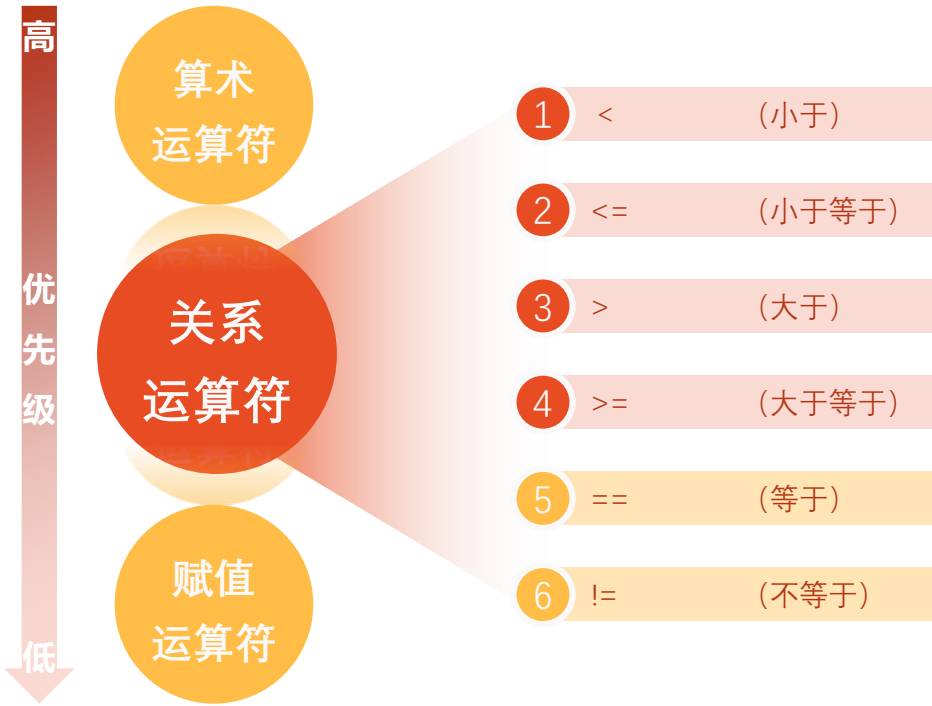
形式3　在else部分又嵌套了多层的if语句

```
if(表达式1)          语句1
else if(表达式2)     语句2
else if(表达式3)     语句3
    ⋮                  ⋮
else if(表达式m)     语句m
else                 语句m+1
```

关系运算符和关系表达式

在C语言中，比较符(或称比较运算符)称为关系运算符。所谓“关系运算”就是“比较运算”，将两个数值进行比较，判断其比较的结果是否符合给定的条件。

关系运算符及其优先次序



- 前 4 种关系运算符的优先级别相同，后2种也相同。前 4 种高于后 2 种。
- 关系运算符的优先级低于算术运算符。
- 关系运算符的优先级高于赋值运算符。

c>a+b 等效于c>(a+b)(关系运算符的优先级低于算术运算符)

a>b==c等效于(a>b)==c(大于运算符>的优先级高于相等运算符==)

a==b<c等效于a==(b<c)(小于运算符<的优先级高于相等运算符==)

a=b>c等效于a=(b>c)(关系运算符的优先级高于赋值运算符)

关系表达式

- 用关系运算符将两个数值或数值表达式连接起来的式子，称为关系表达式。
- 关系表达式的值是一个逻辑值，即“真”或“假”。
- 在C的逻辑运算中，以“1”代表“真”，以“0”代表“假”。

若a=3，b=2，c=1，则：

d=a>b，由于a>b为真，因此关系表达式a>b的值为1，所以赋值后d的值为1。

f=a>b>c，则f的值为0。因为“>”运算符是自左至右的结合方向，先执行“a>b”得值为1，再执行关系运算“1>c”，得值0，赋给f，所以f的值为0

逻辑运算符和逻辑表达式

用逻辑运算符将关系表达式或其他逻辑量连接起来的式子就是逻辑表达式。

逻辑运算符及其优先次序

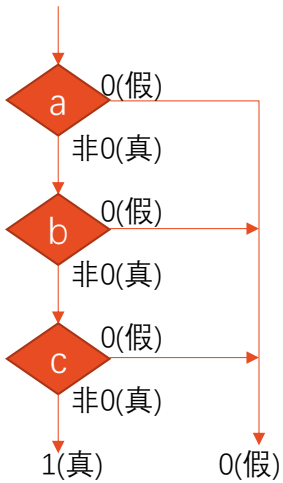
运算符	含义	举例	说明
!	逻辑非(NOT)	!a	如果a为假，则!a为真;如果a为真，则!a为假
&&	逻辑与(AND)	a && b	如果a和b都为真，则结果为真，否则为假
||	逻辑或(OR)	a || b	如果a和b有一个以上为真，则结果为真，二者都为假时，结果为假

- “&&”和“||”是双目运算符，要求有两个运算对象(操作数)；“！”是单目运算符，只要有一个运算对象
- 优先次序：!(非)→＆＆(与)→||(或)，即“！”为三者中最高的；逻辑运算符中的“＆＆”和“||”低于关系运算符，“！”高于算术运算符
- 逻辑运算结果不是0就是1，不可能是其他数值。而在逻辑表达式中作为参加逻辑运算的运算对象可以是0(“假”)或任何非0的数值(按“真”对待)

a	b	!a	!b	a && b	a || b
真(非0)	真(非0)	假(0)	假(0)	真(1)	真(1)
真(非0)	假(0)	假(0)	真(1)	假(0)	真(1)
假(0)	真(非0)	真(1)	假(0)	假(0)	真(1)
假(0)	假(0)	真(1)	真(1)	假(0)	假(0)

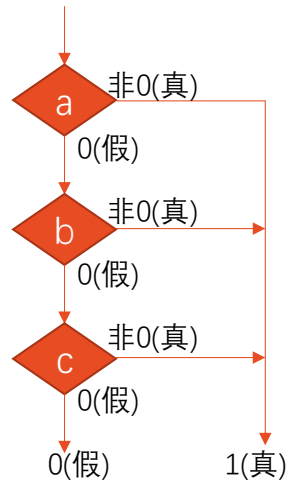
在逻辑表达式的求解中，并不是所有的逻辑运算符都被执行，只是在必须执行下一个逻辑运算符才能求出表达式的解时，才执行该运算符。

- a && b && c。只有a为真(非0)时，才需要判别b的值。只有当a和b都为真时才需要判别c的值。



逻辑表达式

- a || b || c。只要a为真(非0)，就不必判断b和c。只有a为假，才判别b。a和b都为假才判别c。



既然关系表达式和逻辑表达式的值是0和1，而且在判断一个量是否为“真”时，以0代表“假”，以非0代表“真”。那么就可以理解为什么在if语句中表达式可以是任何数值表达式。

```
if (x!=0) 语句1          //括号内的表达式是关系表达式，如果x不等于0，执行语句1
if (x>0 && y>0) 语句2    //表达式是逻辑表达式，如果x和y都大于0，执行语句2
if (x) 语句3             //表达式是变量，如果x不等于0，则条件判断结果为真，执行语句3
if (1) 语句4             //表达式是非0整数，条件判断结果为真，执行语句4
if (0) 语句5             //表达式是整数0,条件判断结果为假，不执行语句5，接着执行下一语句
if(x+3.5) 语句6          //表达式是实数表达式，若x+3.5不等于0，则条件判断结果为真，执行语句6
```

小例子

判别用year表示的某一年是否闰年，可以用一个逻辑表达式来表示。闰年的条件是符合下面二者之一：①能被 4 整除，但不能被100整除，如2008。②能被400整除，如2000。

```
(year % 4 == 0 && year % 100 != 0) || year % 400 == 0
```

条件运算符和条件表达式

```
if (a>b)
    max=a;
else
    max=b;
```

```
max=(a>b) ? a : b;
```

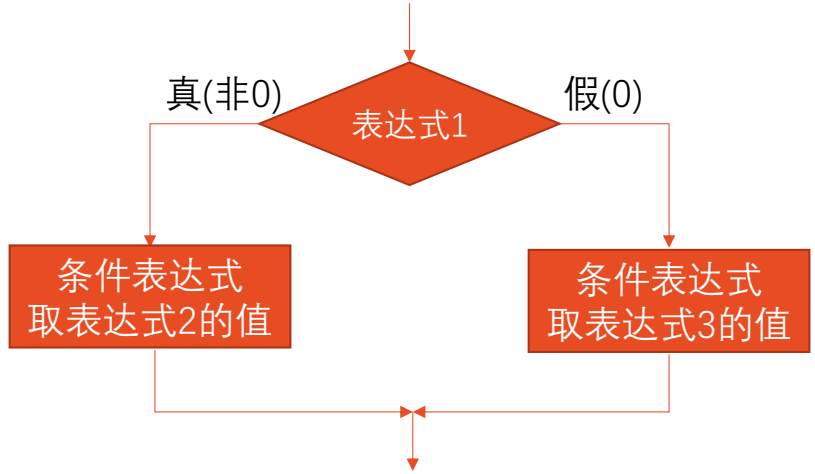
或

```
a>b ? (max=a) : (max=b);  //表达式2和表达式3是赋值表达式
```

表达式1 ? 表达式2 : 表达式3

条件运算符由两个符号(?和:)组成，必须一起使用。要求有3个操作对象，称为三目(元)运算符，它是C语言中唯一的一个三目运算符。

条件运算符的执行顺序：先求解表达式1，若为非0(真)则求解表达式2，此时表达式2的值就作为整个条件表达式的值。若表达式1的值为0(假)，则求解表达式3，表达式3的值就是整个条件表达式的值。

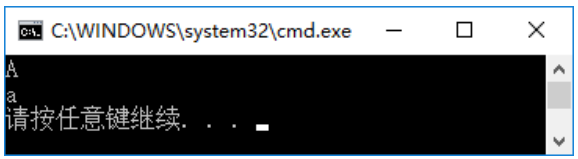


条件运算符和条件表达式

【例4.4】输入一个字符，判别它是否为大写字母，如果是，将它转换成小写字母；如果不是，不转换。然后输出最后得到的字符。

解题思路: 用条件表达式来处理，当字母是大写时，转换成小写字母，否则不转换。

```
#include <stdio.h>
int main()
{
    char ch;
    scanf("%c",&ch);
    ch=(ch>='A'&&ch<='Z')?(ch+32):ch;
    printf("%c\n",ch);
    return 0;
}
```



条件表达式"(ch>='A'&&ch<='Z')?(ch+32):ch"的作用是:如果字符变量ch的值为大写字母，则条件表达式的值为(ch+32)，即相应的小写字母，32是小写字母和大写字母ASCII的差值。如果ch的值不是大写字母，则条件表达式的值为ch，即不进行转换。

选择结构的嵌套

```
if()
        if()  语句1
        else  语句2      内嵌if
else
        if()  语句3
        else  语句4      内嵌if
```

注意

if与else的配对关系。

else总是与它上面的最近的未配对的if配对。

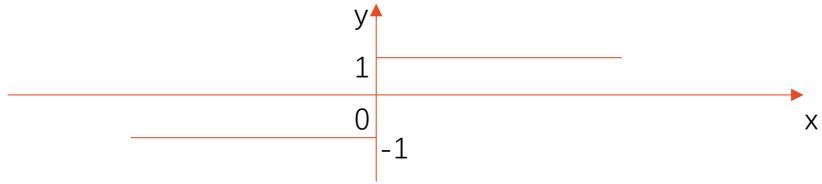
```
if()
    if()  语句1
else
    if()  语句2
else      语句3
```

编程序者把else写在与第1个if(外层if)同一列上，意图是使else与第1个if对应，但实际上else是与第2个if配对，因为它们相距最近。

如果if与else的数目不一样，为实现程序设计者的思想,可以加花括号来确定配对关系。

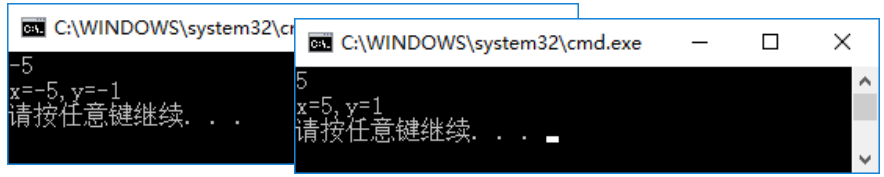
```
if()
{
    if()  语句1      内嵌if
}
else      语句2
```

条件运算符和条件表达式



【例4.5】有一阶跃函数：$y=\begin{cases}-1 & (x<0)\\ 0 & (x=0)\\ 1 & (x>0)\end{cases}$

编一程序,输入一个x值，要求输出相应的y值。



算法步骤

先后用3个独立的if语句处理

S1：输入x

S2：若x<0,则y =-1

S3：若x=0,则y=0

S4：若x>0,则y=1

S5：输出y

```
#include <stdio.h>
int main()
{
    int x,y;
    scanf("%d",&x);
    if(x<0)
        y=-1;
    else
        if(x==0) y=0;
        else y=1;
        printf("x=%d,y=%d\n",x,y);
    return 0;
}
```

算法步骤

用一个嵌套的if语句处理

S1：输入x

S2：若x<0,则y=-1

S3：否则

S4：若x=0,则y=0

S5：否则(即x>0),则y=1

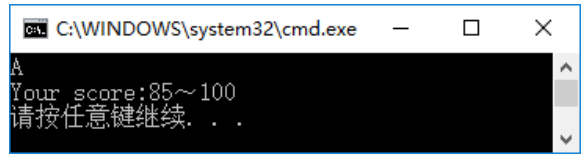
S6：输出y

```
#include <stdio.h>
int main()
{
    int x,y;
    scanf("%d",&x);
    if(x>=0)//
        if(x>0) y=1;
        else    y=0;
    else        y=-1;
    printf("x=%d,y=%d\n",x,y);
    return 0;
}
```

用switch语句实现多分支选择结构

【例4.6】要求按照考试成绩的等级输出百分制分数段，A等为85分以上，B等为70～84分，C等为60～69分，D等为 60分以下。成绩的等级由键盘输入。

```
#include <stdio.h>
int main()
{
    char grade;
    scanf("%c",&grade);
    printf("Your score:");
    switch(grade)
    {
        case 'A': printf("85 ~ 100\n");break;
        case 'B': printf("70 ~ 84\n");break;
        case 'C': printf("60 ~ 69\n");break;
        case 'D': printf("<60\n");break;
        default:  printf("enter data error!\n");
    }
    return 0;
}
```



等级grade定义为字符变量，从键盘输入一个大写字母，赋给变量grade，switch得到grade的值并把它和各case中给定的值('A','B','C','D'之一)相比较，如果和其中之一相同(称为匹配)，则执行该case后面的语句(即printf语句)。如果输入的字符与'A','B','C','D'都不相同，就执行default后面的语句，

注意在每个case后面后的语句中，最后都有一个break语句，它的作用是使流程转到switch语句的末尾(即右花括号处)。

用switch语句实现多分支选择结构

```
switch(表达式)
{
    case  常量1 : 语句1
    case  常量2 : 语句2
      ⋮     ⋮     ⋮
    case  常量n : 语句n
    default :     语句n+1
}
```

(1) 括号内的“表达式”，其值的类型应为整数类型(包括字符型)。

(2) 花括号内是一个复合语句，内包含多个以关键字case开头的语句行和最多一个以default开头的行。case后面跟一个常量(或常量表达式)，它们和default都是起标号作用，用来标志一个位置。执行switch语句时，先计算switch后面的“表达式”的值，然后将它与各case标号比较，如果与某一个case标号中的常量相同，流程就转到此case标号后面的语句。如果没有与switch表达式相匹配的case常量，流程转去执行default标号后面的语句。

(3) 可以没有default标号，此时如果没有与switch表达式相匹配的case常量，则不执行任何语句。

(4) 各个case标号出现次序不影响执行结果。

(5) 每一个case常量必须互不相同；否则就会出现互相矛盾的现象。

(6) case标号只起标记的作用。在执行switch语句时，根据switch表达式的值找到匹配的入口标号，在执行完一个case标号后面的语句后，就从此标号开始执行下去，不再进行判断。因此，一般情况下，在执行一个case子句后，应当用break语句使流程跳出switch结构。最后一个case子句(今为default子句)中可不加break语句。

(7) 在case子句中虽然包含了一个以上执行语句，但可以不必用花括号括起来，会自动顺序执行本case标号后面所有的语句。当然加上花括号也可以。

(8) 多个case标号可以共用一组执行语句。

用switch语句实现多分支选择结构

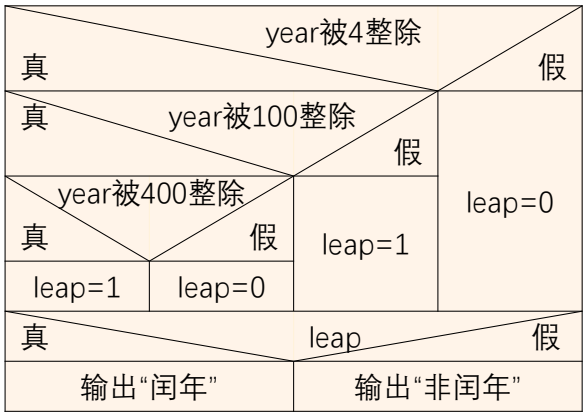
【例4.7】用switch语句处理菜单命令。在许多应用程序中，用菜单对流程进行控制，例如从键盘输入一个'A'或'a'字符，就会执行A操作，输入一个'B'或'b'字符，就会执行B操作。

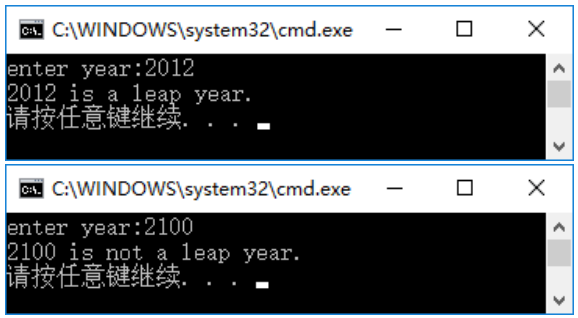
```
#include <stdio.h>
int main()
{     void action1(int,int),action2(int,int);     //函数声明
      char ch;
      int a=15,b=23;
      ch=getchar();
      switch(ch)
      {     case 'a':
            case 'A': action1(a,b);break;          //调用action1函数，执行A操作
            case 'b':
            case 'B': action2(a,b);break;          //调用action2函数，执行B操作
                  ⋮
            default:  putchar('\a');               //如果输入其他字符，发出警告
      }
      return 0;
}
```

```
void action1(int x,int y)        //执行加法的函数
{     printf("x+y=%d\n",x+y);
}

void action2(int x,int y)        //执行乘法的函数
{     printf("x*y=%d\n",x*y);
}
```

选择结构程序综合举例

【例4.8】写一程序，判断某一年是否为闰年。





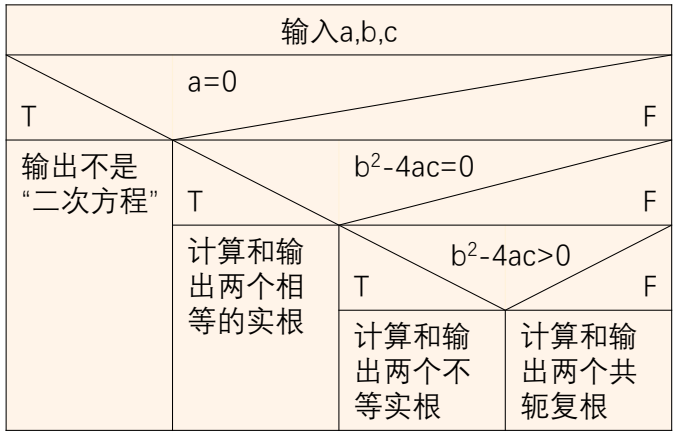

```
#include <stdio.h>
int main()
{
    int year,leap;
    printf("enter year:");
    scanf("%d",&year);
    if(year%4==0)
    {
        if(year%100==0)
        {
            if(year%400==0)
                leap=1;
            else
                leap=0;
        }
        else
            leap=1;
    }
    else
        leap=0;
    if(leap)
        printf(
    else
        printf(
    printf("a lea
    return 0;
}
```

```
if(year%4!=0)
    leap=0;
else if (year%100!=0)
    leap=1;
else if(year%400!=0)
    leap=0;
else
    leap=1;
```

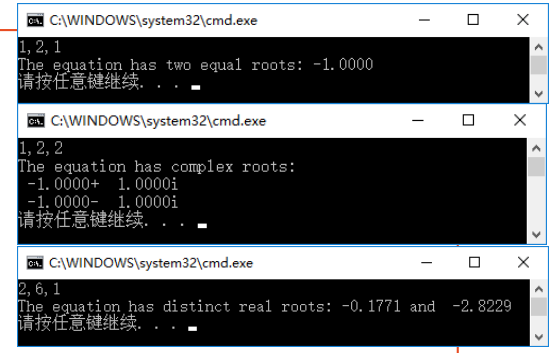
```
if((year%4==0 && year%100!=0) || (year%400==0))
  leap=1;
else
  leap=0;
```

选择结构程序综合举例

【例4.9】求$ax^2+bx+c=0$方程的解。




```
#include <stdio.h>
#include <math.h>
int main()
{     double a,b,c,disc,x1,x2,realpart,imagpart;
      scanf("%lf,%lf,%lf",&a,&b,&c);
      printf("The equation ");
      if(fabs(a)<=1e-6)
            printf("is not a quadratic\n");
      else
      {     disc=b*b-4*a*c;
            if(fabs(disc)<=1e-6)
                  printf("has two equal roots:%8.4f\n",-b/(2*a));
            else
                  if(disc>1e-6)
                  {     x1=(-b+sqrt(disc))/(2*a);
                        x2=(-b-sqrt(disc))/(2*a);
                        printf("has distinct real roots:%8.4f and %8.4f\n",x1,x2);
                  }
                  else
                  {     realpart=-b/(2*a);                  //realpart是复根的实部
                        imagpart=sqrt(-disc)/(2*a);   //imagpart是复根的虚部
                        printf("has complex roots:\n");
                        printf("%8.4f+%8.4fi\n",realpart,imagpart);    //输出一个复数
                        printf("%8.4f-%8.4fi\n",realpart,imagpart);    //输出另一个复数
                  }
      }
      return 0;
}
```



选择结构程序综合举例

【例4.10】运输公司对用户计算运输费用。路程越远，运费越低。标准如下:

s<250没有折扣
250≤s< 5002%折扣
500≤s< 10005%折扣
1000≤s< 20008%折扣
2000≤s< 300010%折扣
3000≤s15%折扣

p：每吨每千米货物的基本运费
w：货物重量
s：运输距离
d：折扣
f：总运费

f=p*w*s*(1-d)

```
#include <stdio.h>
int main()
{    int c,s;
     float p,w,d,f;
     printf("please enter price,weight,discount:");//提示输入的数据
     scanf("%f,%f,%d",&p,&w,&s);          //输入单价、重量、距离
     if(s>=3000) c=12;  //3000km以上为同一折扣
     else c=s/250;//3000km以下各段折扣不同，c的值不相同
     switch(c)
     {    case 0: d=0;break;              //c=0,代表250km以下,折扣d=0
          case 1: d=2;break;              //c=1,代表250 ~ 500km以下,折扣d=2%
          case 2:
          case 3: d=5;break;              //c=2和3,代表500 ~ 1000km,折扣d=5%
          case 4:
          case 5:
          case 6:
          case 7: d=8;break;              //c=4 ~ 7,代表1000 ~ 2000km,折扣d=8%
          case 8:
          case 9:
          case 10:
          case 11: d=10;break;            //c=8 ~ 11,代表2000 ~ 3000km,折扣d=10%
          case 12: d=15;break;            //c12,代表3000km以上,折扣d=15%
     }
     f=p*w*s*(1-d/100);                   //计算总运费
     printf("freight=%10.2f\n",f);        //输出总运费，取两位小数
     return 0;
}
```

```
C:\WINDOWS\system32\cmd.exe
please enter price,weight,discount:100,20,300
freight= 588000.00
请按任意键继续. . .
```

第5章

循环结构程序设计

为什么需要循环控制

- 要向计算机输入全班50个学生的成绩；
  (重复50次相同的输入操作)
- 分别统计全班50个学生的平均成绩；
  (重复50次相同的计算操作)

解决方法

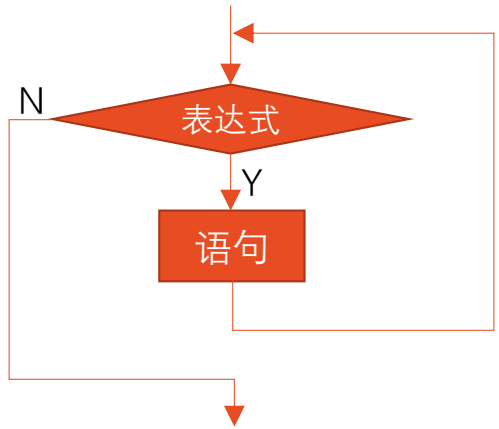
```
scanf("%f,%f,%f,%f,%f",&score1,&score2,&score3,&score4,&score5);
//输入一个学生5门课的成绩
aver=(score1+score2+score3+score4+score5)/5;
//求该学生平均成绩
printf("aver=%7.2f",aver);
//输出该学生平均成绩
```

重复写49个同样的程序段 


```
i=1;              //设整型变量i初值为1
while(i<=50) //当i的值小于或等于50时执行花括号内的语句
{    scanf("%f,%f,%f,%f,%f",&score1,&score2,&score3,&score4,&score5);
     aver=(score1+score2+score3+score4+score5)/5;
     printf("aver=%7.2f",aver);
     i++;      //每执行完一次循环使i的值加1
}
```

用while语句实现循环

while(表达式) 语句



while语句可简单地记为：只要当循环条件表达式为真(即给定的条件成立)，就执行循环体语句。

“语句”就是循环体。循环体可以是一个简单的语句，可以是复合语句(用花括号括起来的若干语句)。

执行循环体的次数是由循环条件控制的，这个循环条件就是上面一般形式中的“表达式”，它也称为循环条件表达式。当此表达式的值为“真”(以非0值表示)时，就执行循环体语句；为“假”(以0表示)时，就不执行循环体语句。

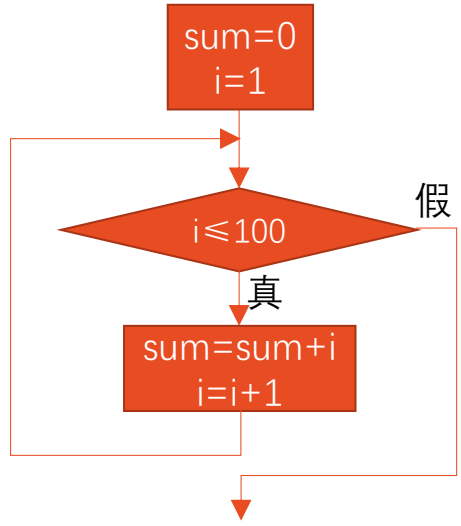
注意

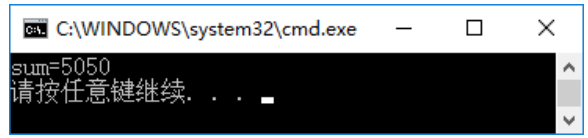
while循环的特点是先判断条件表达式，后执行循环体语句。

while语句实现循环

【例5.1】求1+2+3+...+100，即$\sum_{n=1}^{100} n$

```
#include<stdio.h>
int main()
{
    int i=1,sum=0;              //定义变量i的初值为1,sum的初值为0
    while(i<=100)               //当i>100，条件表达式i<=100的值为假，不执行循环体
    {                           //循环体开始
        sum=sum+i;              //第1次累加后，sum的值为1
        i++;                    //加完后，i的值加1，为下次累加做准备
    }                           //循环体结束
    printf("sum=%d\n",sum);     //输出1+2+3…+100的累加和
    return 0;
}
```

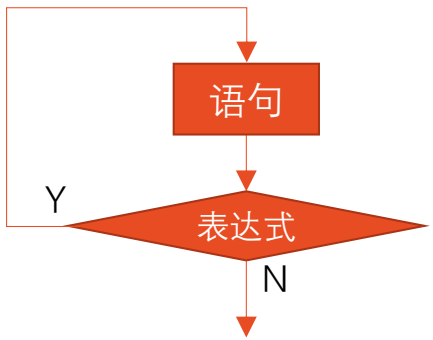




(1) 循环体如果包含一个以上的语句，应该用花括号括起来，作为复合语句出现。
(2) 不要忽略给i和sum**赋初值**，否则它们的值是不可预测的，结果显然不正确。
(3) 在循环体中应有使循环趋向于结束的语句。如本例中的"i++；"语句。如果无此语句，则i的值始终不改变，循环永远不结束。

用do…while语句实现循环

```
do
    语句
while(表达式);
```



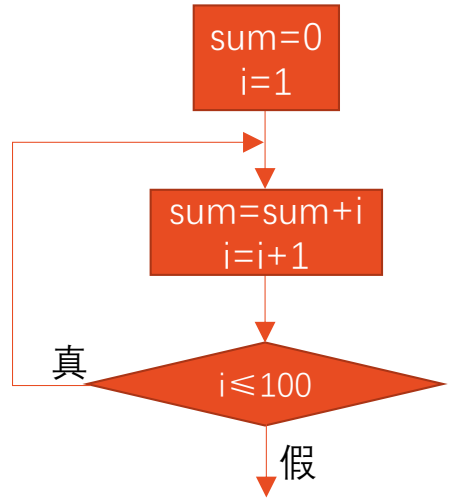
注意

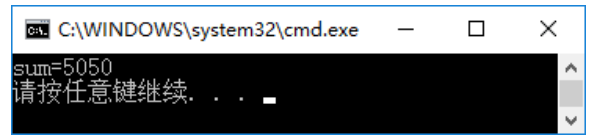
do…while语句的特点是，先无条件地执行循环体，然后判断循环条件是否成立。

用do…while语句实现循环

【例5.2】用do...while语句求1+2+3+...+100，即$\sum_{n=1}^{100} n$

```
#include <stdio.h>
int main()
{
    int i=1,sum=0;
    do
    {
        sum=sum+i;
        i++;
    }while(i<=100);
    printf("sum=%d\n",sum);
    return 0;
}
```





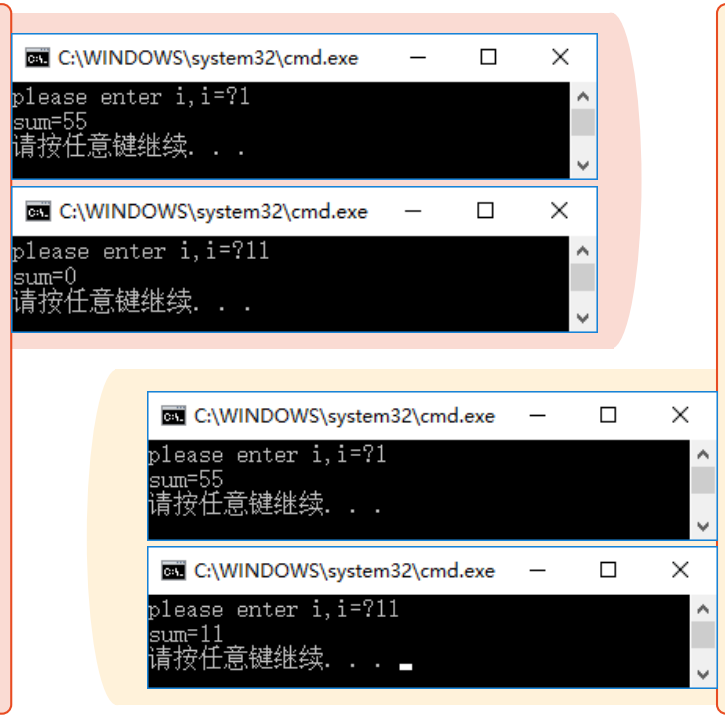
在一般情况下，用while语句和用do…while语句处理同一问题时，若二者的循环体部分是一样的，那么结果也一样。

但是如果while后面的表达式一开始就为假(0值)时，两种循环的结果是不同的。

用do…while语句实现循环

【例5.3】while和do...while循环的比较。

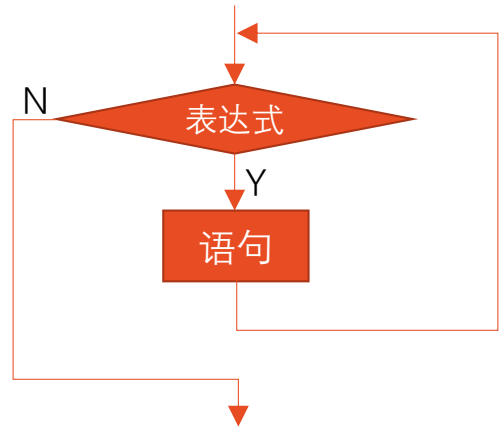
```
#include <stdio.h>
int main()
{
    int i,sum=0;
    printf("please enter i,i=?");
    scanf("%d",&i);
    while(i<=10)
    {
        sum=sum+i;
        i++;
    }
    printf("sum=%d\n",sum);
    return 0;
}
```




```
#include <stdio.h>
int main()
{
    int i,sum=0;
    printf("please enter i,i=?");
    scanf("%d",&i);
    do
    {
        sum=sum+i;
        i++;
    }while(i<=10);
    printf("sum=%d\n",sum);
    return 0;
}
```

用for语句实现循环

for(表达式1；表达式2；表达式3)
语句



表达式1: 设置初始条件，只执行一次。可以为零个、一个或多个变量设置初值。

表达式2: 是循环条件表达式，用来判定是否继续循环。在每次执行循环体前先执行此表达式，决定是否继续执行循环。

表达式3: 作为循环的调整，例如使循环变量增值，它是在执行完循环体后才进行的。

注意

while循环的特点是先判断条件表达式，后执行循环体语句。

用for语句实现循环

for(表达式1；表达式2；表达式3)
**　　语句**

for(循环变量赋值；表达式2；表达式3)
**　　语句**

for语句更为灵活，不仅可以用于循环次数已经确定的情况，还可以用于循环次数不确定而只给出循环结束条件的情况，它完全可以代替while语句。

表达式1: 设置初始条件，只执行一次。可以为零个、一个或多个变量设置初值。

表达式2: 是循环条件表达式，用来判定是否继续循环。在每次执行循环体前先执行此表达式，决定是否继续执行循环。

表达式3: 作为循环的调整，例如使循环变量增值，它是在执行完循环体后才进行的。

用for语句实现循环

for(表达式1；表达式2；表达式3)
**　　语句**

≡

```
表达式1;
while 表达式2
{
　　语句
　　表达式3
}
```

for语句的执行过程如下:

(1) 求解表达式1。

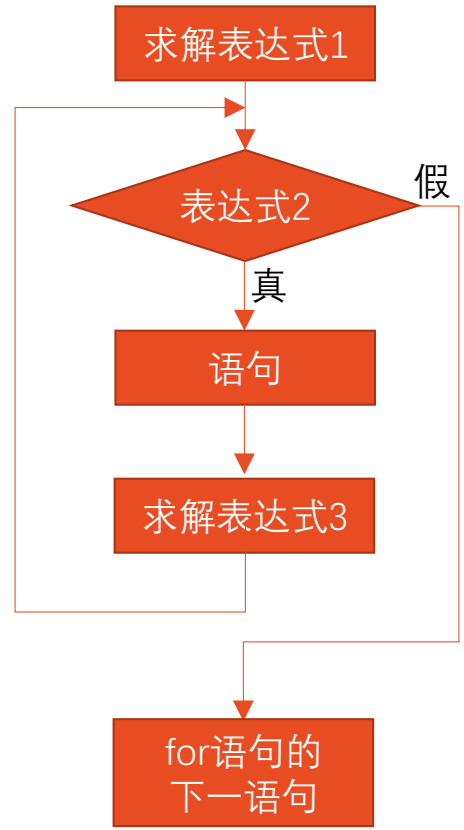
(2) 求解表达式2，若此条件表达式的值为真(非0)，则执行for语句中的循环体，然后执行第(3)步。若为假(0)，则结束循环，转到第(5)步。

(3) 求解表达式3。

(4) 转回步骤(2)继续执行。

注意: 在执行完循环体后，循环变量的值“超过”循环终值，循环结束。

(5) 循环结束，执行for语句下面的一个语句。



用for语句实现循环

for(表达式1；表达式2；表达式3）
语句

注意

- “表达式1”可以省略，即不设置初值，但表达式1后的分号不能省略。例如: for(； i<=100;i++)。应当注意: 由于省略了表达式1，没有对循环变量赋初值，因此，为了能正常执行循环，应在for语句之前给循环变量赋以初值。
- 表达式2也可以省略，即不用表达式2来作为循环条件表达式，不设置和检查循环的条件。此时循环无终止地进行下去，也就是认为表达式2始终为真。
- 表达式3也可以省略，但此时程序设计者应另外设法保证循环能正常结束。
- 甚至可以将3个表达式都可省略，即不设初值，不判断条件(认为表达式2为真值)，循环变量也不增值，无终止地执行循环体语句，显然这是没有实用价值的。
- 表达式1可以是设置循环变量初值的赋值表达式，也可以是与循环变量无关的其他表达式。表达式3也可以是与循环控制无关的任意表达式。但不论怎样写for语句，都必须使循环能正常执行。
- 表达式1和表达式3可以是一个简单的表达式，也可以是逗号表达式，即包含一个以上的简单表达式，中间用逗号间隔。
- 表达式2一般是关系表达式或逻辑表达式，但也可以是数值表达式或字符表达式，只要其值为非零，就执行循环体。
- for语句的循环体可为空语句，把本来要在循环体内处理的内容放在表达式3中，作用是一样的。可见for语句功能强，可以在表达式中完成本来应在循环体内完成的操作。
- C 99允许在for语句的“表达式1”中定义变量并赋初值。

循环的嵌套

01

```
while()
{   ⋮
    while()
    {…}
}
```

内层循环

02

```
do
{   ⋮
    do
    {…}
    while();
}while();
```

内层循环

03

```
for( ; ; )
{
    for( ; ; )
    {…}
}
```

内层循环

04

```
while()
{   ⋮
    do
    {…}
    while();
    ⋮
}
```

内层循环

05

```
for( ; ; )
{   ⋮
    while()
    {…}
    ⋮
}
```

内层循环

06

```
do
{   ⋮
    for( ; ; )
    {…}
}while();
```

内层循环

几种循环的比较

(1) 3种循环都可以用来处理同一问题，一般情况下它们可以互相代替。

(2) 在while循环和do…while循环中，只在while后面的括号内指定循环条件，因此为了使循环能正常结束，应在循环体中包含使循环趋于结束的语句(如i++，或i=i+1等)。

for循环可以在表达式3中包含使循环趋于结束的操作，甚至可以将循环体中的操作全部放到表达式3中。因此for语句的功能更强，凡用while循环能完成的，用for循环都能实现。

(3) 用while和do…while循环时，循环变量初始化的操作应在while和do…while语句之前完成。而for语句可以在表达式1中实现循环变量的初始化。

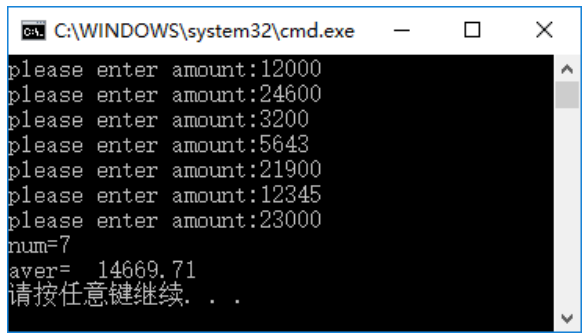
(4) while循环、do…while循环和for循环都可以用break语句跳出循环，用continue语句结束本次循环。


改变循
环执行
的状态
break
continue

用break语句提前终止循环

【例5.4】在全系1000名学生中举行慈善募捐，当总数达到10万元时就结束，统计此时捐款的人数以及平均每人捐款的数目。

```
#include <stdio.h>
#define SUM 100000    //指定符号常量SUM代表10万
int main()
{
    float amount,aver,total;
    int i;
    for (i=1,total=0;i<=1000;i++)
    {
        printf("please enter amount:");
        scanf("%f",&amount);
        total=total+amount;
        if(total>=SUM) break;
    }
    aver=total/i;
    printf("num=%d\naver=%10.2f\n",i,aver);
    return 0;
}
```



for语句指定执行循环体1000次。每次循环中，输入一个捐款人的捐款数，并累加到total中。设置了if语句，在每一次累加捐款数amount后，立即检查累加和total是否达到或超过SUM(即100 000)，若超过就执行break语句，流程跳转到循环体的花括号外，提前结束循环。

用break语句提前终止循环

```
break;
```

作用：使流程跳到循环体之外，接着执行循环体下面的语句。

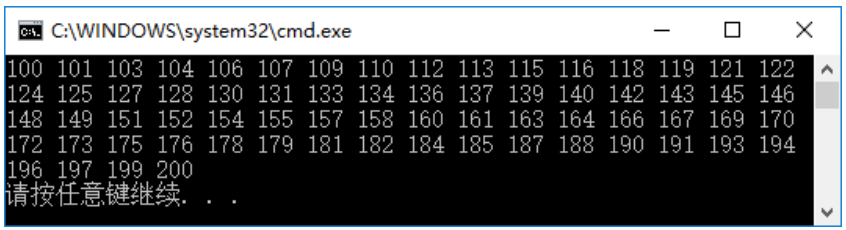
注意：break语句只能用于循环语句和switch语句之中，而不能单独使用。

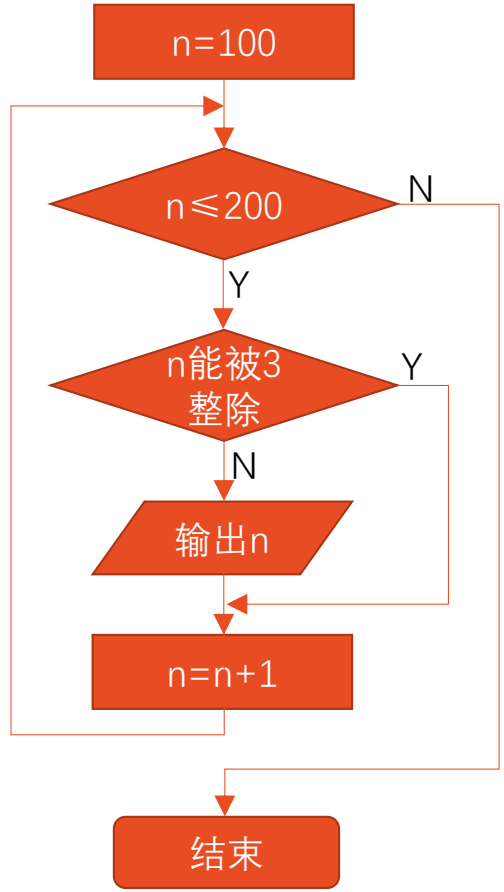
用continue语句提前结束本次循环

【例5.5】要求输出100～200之间的不能被3整除的数。

```
#include <stdio.h>
int main()
{     int n;
      for (n=100;n<=200;n++)
      {     if (n%3==0)
                  continue;
            printf("%d ",n);
      }
      printf("\n");
      return 0;
}
```

当n能被3整除时，执行continue语句，流程跳转到表示循环体结束的右花括号的前面(注意不是右花括号的后面)，从而跳过printf函数语句，结束本次循环，然后进行循环变量的增值(n++)，只要n<=200，就会接着执行下一次循环。





用continue语句提前结束本次循环

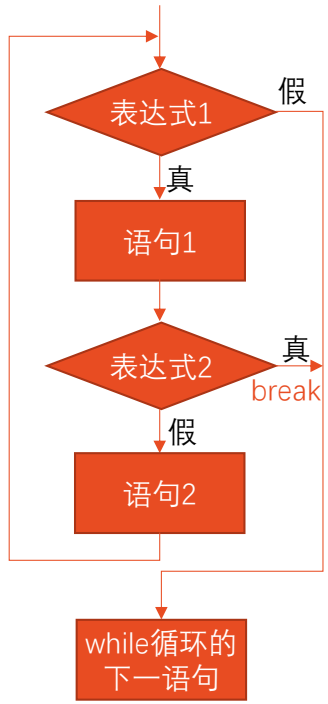
```
continue;
```

作用：结束本次循环，即跳过循环体中下面尚未执行的语句，转到循环体结束点之前，接着执行for语句中的“表达式3”，然后进行下一次是否执行循环的判定。

break语句和continue语句的区别

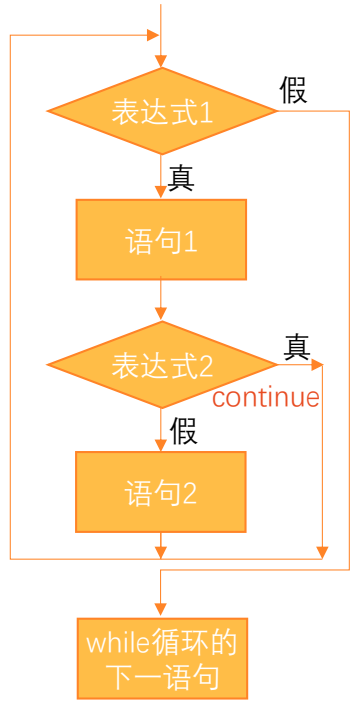
break;

```
while(表达式1)
{
        语句1
        if(表达式2) break;
        语句2
}
```



continue;

```
while(表达式1)
{
        语句1
        if(表达式2) continue;
        语句2
}
```



continue语句只结束本次循环，而非终止整个循环。break语句结束整个循环，不再判断执行循环的条件是否成立。

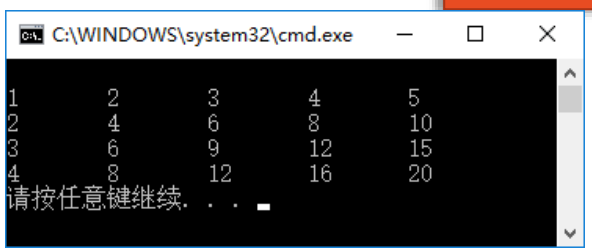
break语句和continue语句的区别

【例5.6】输出以下4×5的矩阵。

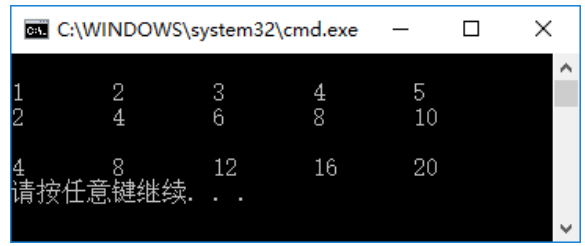
1	2	3	4	5
2	4	6	8	10
3	6	9	12	15
4	8	12	16	20

```
#include <stdio.h>
int main()
{
    int i,j,n=0;
    for(i=1;i<=4;i++)
        for(j=1;j<=5;j++,n++)          //n用来累计输出数据的个数
        {   if(n%5==0) printf("\n");   //控制在输出5个数据后换行
            printf("%d\t",i*j);
        }
    printf("\n");
    return 0;
}
```

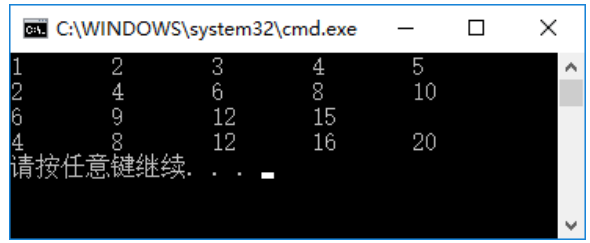
本程序包括一个双重循环，是for循环的嵌套。外循环变量i由1变到4，用来控制输出4行数据;内循环变量j由1变到5，用来控制输出每行中的5个数据。



if (i==3 && j==1) break;



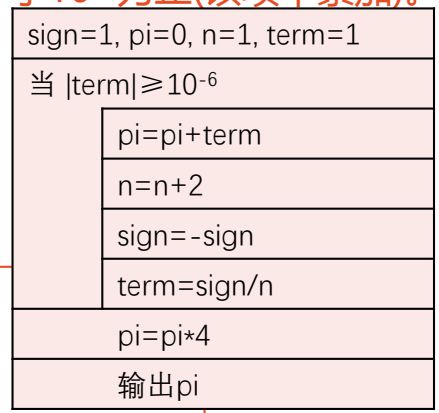
if (i==3 && j==1) continue;



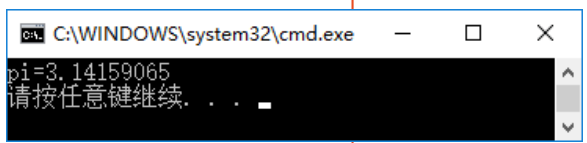
循环程序举例

【例5.7】用公式$\frac{\pi}{4} \approx 1-\frac{1}{3}+\frac{1}{5}-\frac{1}{7}+\cdots$求π的近似值，直到发现某一项的绝对值小于$10^{-6}$为止(该项不累加)。

解题思路： 找规律：
(1) 每项的分子都是1。
(2) 后一项的分母是前一项的分母加2。
(3) 第1项的符号为正，从第2项起，每一项的符号与前一项的符号相反。
在每求出一项后，检查它的绝对值是否大于或等于10^{-6}。




```
#include <stdio.h>
#include <math.h>                    //程序中用到数学函数fabs，应包含头文件math.h
int main()
{    int sign=1;                     //sign用来表示数值的符号
     double pi=0.0,n=1.0,term=1.0;   //pi开始代表多项式的值，最后代表π的值, n代表分母，term代表当前项的值
     while(fabs(term)>=1e-6)         //检查当前项term的绝对值是否大于或等于10-6
     {    pi=pi+term;                //把当前项term累加到pi中
          n=n+2;                     //n+2是下一项的分母
          sign=-sign;                //sign代表符号，下一项的符号与上一项符号相反
          term=sign/n;               //求出下一项的值term
     }
     pi=pi*4;                        //多项式的和pi乘以4，才是π的近似值
     printf("pi=%10.8f\n",pi);       //输出π的近似值
     return 0;
}
```



循环程序举例

【例5.8】求Fibonacci(斐波那契)数列的前40个数。这个数列有如下特点: 第1，2两个数为1，1。从第3个数开始，该数是其前面两个数之和。即该数列为1,1,2,3,5,8,13,...,用数学方式表示为:

$$\begin{cases} F_1 = 1 & (n = 1) \\ F_2 = 1 & (n = 2) \\ F_n = F_{n-1} + F_{n-2} & (n \geq 3) \end{cases}$$

这是一个有趣的古典数学问题: 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3个月后每个月又生一对兔子。假设所有兔子都不死，问每个月的兔子总数为多少?

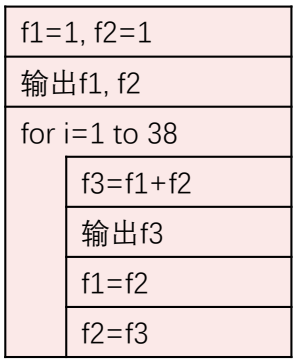
兔子繁殖的规律

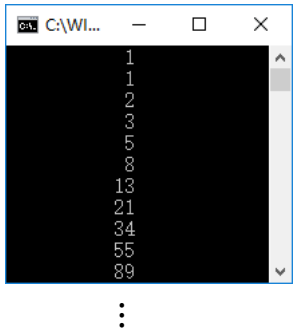
月数	小兔子对数	中兔子对数	老兔子对数	兔子总对数
1	1	0	0	1
2	0	1	0	1
3	1	0	1	2
4	1	1	1	3
5	2	1	2	5
6	3	2	3	8
7	5	3	5	13
⋮	⋮	⋮	⋮	⋮

注: 假设不满1个月的为小兔子，
满1个月不满2个月的为中兔子，
满2个月以上的为老兔子。

循环程序举例

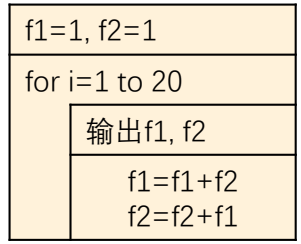
【例5.8】求Fibonacci(斐波那契)数列的前40个数。

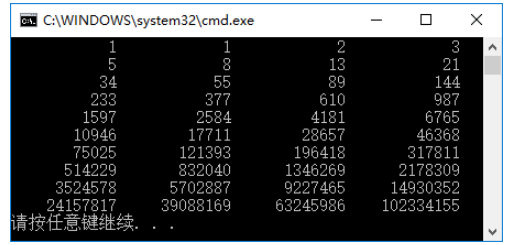




⋮

```
#include <stdio.h>
int main()
{
        int f1=1,f2=1,f3;
        int i;
        printf("%12d\n%12d\n",f1,f2);
        for(i=1; i<=38; i++)
        {
                f3=f1+f2;
                printf("%12d\n",f3);
                f1=f2;
                f2=f3;
        }
        return 0;
}
```



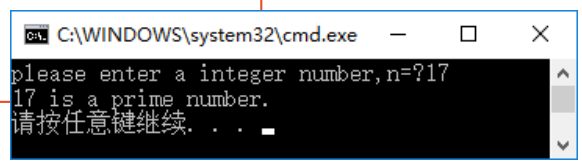


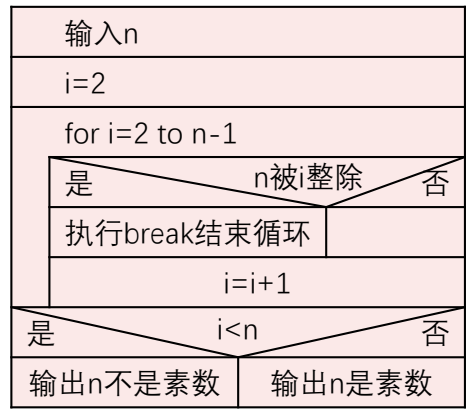

```
#include <stdio.h>
int main()
{
        int f1=1,f2=1;
        int i;
        for(i=1; i<=20; i++)      //每个循环输出2个月的数据，故只需循环20次
        {
                printf("%12d %12d ",f1,f2);     //输出已知的两个月的兔子数
                if(i%2==0) printf("\n");
                f1=f1+f2;               //计算出下一个月的兔子数，并存放在f1中
                f2=f2+f1;               //计算出下两个月的兔子数，并存放在f2中
        }
        return 0;
}
```

循环程序举例

【例5.9】输入一个大于3的整数n，判定它是否为素数(prime，又称质数)。

```
#include <stdio.h>
int main()
{     int n,i;
      printf("please enter a integer number,n=?");
      scanf("%d",&n);
      for (i=2;i<n;i++)
            if(n%i==0) break;
      if(i<n) printf("%d is not a prime number.\n",n);
      else printf("%d is a prime number.\n",n);
      return 0;
}
```





若n能被2～(n-1)之间的一个整数整除，则执行break语句，提前结束循环，流程跳转到循环体之外。此时i<n。如果n不能被2～(n-1)之间任何的一个整数整除，则不会执行break语句，循环变量i一直变化到等于n，然后由第1个判断框判定“i<n”条件不成立，从而结束循环。这种正常结束的循环，其循环变量的值必然大于事先指定的循环变量终值(本例中循环变量终值为n-1)。
因此，只要在循环结束后检查循环变量i的值，就能判定循环是提前结束还是正常结束的。从而判定n是否为素数。
希望读者理解和掌握这一方法，以后会常用到。

循环程序举例

【例5.9】输入一个大于3的整数n，判定它是否为素数(prime，又称质数)。

```
#include <stdio.h>
int main()
{     int n,i;
      printf("please enter a integer number,n=?");
      scanf("%d",&n);
      for (i=2;i<n;i++)
            if(n%i==0) break;
      if(i<n) printf("%d is not a prime number.\n",n);
      else printf("%d is a prime number.\n",n);
      return 0;
}
```

程序改进：

其实n不必被2～(n-1)范围内的各整数去除，只须将n被2～$\sqrt{n}$之间的整数除即可。因为n的每一对因子，必然有一个小于n,另一个大于n。

```
#include <stdio.h>
#include <math.h>
int main()
{     int n,i,k;
      printf("please enter a integer number,n=?");
      scanf("%d",&n);
      k=sqrt(n);
      for (i=2;i<=k;i++)
            if(n%i==0) break;
      if(i<=k) printf("%d is not a prime number.\n",n);
      else printf("%d is a prime number.\n",n);
      return 0;
}
```

循环程序举例

【例5.9】输入一个大于3的整数n，判定它是否为素数(prime，又称质数)。

其他求素数方法

```
for(t=1,i=2; i<=(int)sqrt(n); i++)   //先定义t为int型，t作为标志变量
      if(n%i==0)
            t=0;                    //t=0表示n能被i整除，n不是素数
if(t)                               //如果t=1表示n是素数
      printf("%d is prime.\n",n);
```

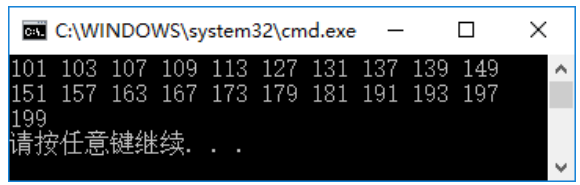
```
for(t=1,i=2; i<=(int)sqrt(n); i++)
      if(n%i==0){
            t=0;
            break;
      }
if(t)
      printf("%d is prime.\n",n);
```

```
for(t=1,i=2; i<=sqrt(n) && t; i++)
      if(n%i==0)
            t=0;
if(t)
      printf("%d is prime.\n",n);
```

循环程序举例

【例5.10】求100～200间的全部素数。

```
#include<stdio.h>
#include<math.h>
int main()
{     int n,k,i,m=0;
      for(n=101;n<=200;n=n+2)          //n从100变化到200，对每个奇数n进行判定
      {     k=sqrt(n);
            for(i=2;i<=k;i++)
            if(n%i==0) break;          //如果n被i整除，终止内循环，此时i<k+1
            if(i>=k+1)                 //若i>=k+1，表示n未曾被整除
            {     printf("%d ",n);     //应确定n是素数
                  m=m+1;               //m用来控制换行，一行内输出10个素数
            }
            if(m%10==0) printf("\n");      //m累计到10的倍数，换行
      }
      printf ("\n");
      return 0;
}
```



循环程序举例

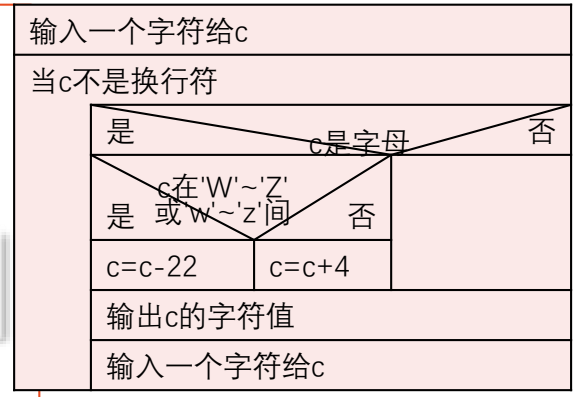
【例5.11】译密码。为使电文保密，往往按一定规律将其转换成密码，收报人再按约定的规律将其译回原文。例如，可以按以下规律将电文变成密码:将字母A变成字母E，a变成e，即变成其后的第4个字母，W变成A，X变成B，Y变成C，Z变成D。

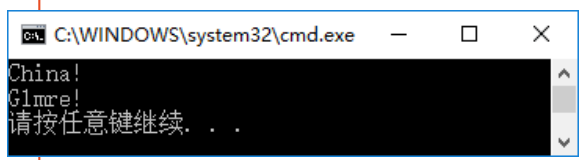
解题思路:

(1) 判断哪些字符不需要改变，哪些字符需要改变。

(2)通过改变字符c的ASCII值的方式将其变为指定的字母。'A'~'V'或'a'~'v'： c=c+4；'W'~'Z'或'w'~'z'： c=c-22。

```
#include <stdio.h>
int main()
{    char c;
     while((c=getchar()!='\n')      //输入一个字符给字符变量c
     {    if((c>='a' && c<='z') || (c>='A' && c<='Z'))      //检查c的值是否为换行符'\n'
          {    c=c+4       //只要是字母，都先加4
               //如果是26个字母中最后4个字母之一， c值变为对应的最前面的4个字母
               if(c>'Z' && c<='Z'+4 || c>'z' && c<='z'+4)  c=c-26;
          }
          printf("%c",c);              //输出已改变的字符
                                       //再输入下一个字符给字符变量c
     }
     printf("\n");
     return 0;
}
```





第 6 章

利用数组处理批量数据

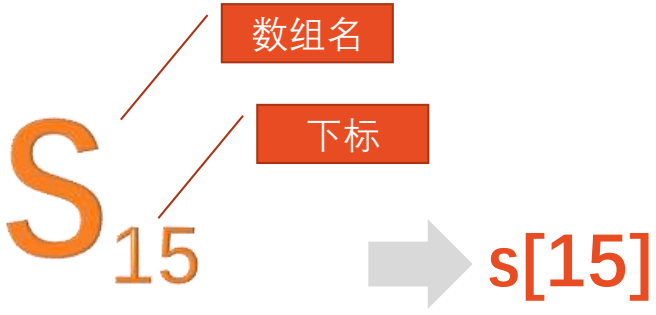
为什么需要循环控制

- **要向计算机输入全班50个学生一门课程的成绩**

解决方法

用50个float型简单变量表示学生的成绩

- **烦琐，**如果有1000名学生怎么办呢？
- **没有反映出这些数据间的内在联系，**实际上这些数据是同一个班级、同一门课程的成绩，它们具有相同的属性。



数组

(1) 数组是一组有序数据的集合。数组中各数据的排列是有一定规律的，下标代表数据在数组中的序号。

(2) 用数组名和下标即可唯一地确定数组中的元素。

(3) 数组中的每一个元素都属于同一个数据类型。

定义一维数组

类型说明符 数组名[常量表达式]

(1) 数组名的命名规则和变量名相同，遵循标识符命名规则。

(2) 在定义数组时，需要指定数组中元素的个数，方括号中的常量表达式用来表示元素的个数，即数组长度。

(3) 常量表达式中可以包括常量和符号常量，不能包含变量。

相当于定义了10个简单的整型变量

注意

- 数组元素的**下标从0开始**，用“int a[10];”定义数组，则最大下标值为9，不存在数组元素a[10]

引用一维数组元素

数组名[下标]

只能引用数组元素而不能一次整体调用整个数组全部元素的值。

数组元素与一个简单变量的地位和作用相似。

“下标”可以是整型常量或整型表达式。

注意

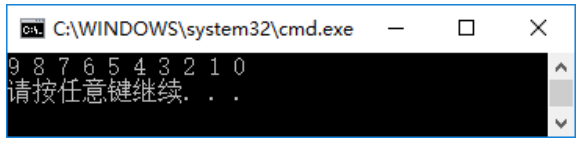
- 定义数组时用到的“数组名[常量表达式]” 和引用数组元素时用的“数组名[下标]”形式相同，但含义不同。

```
int a[10];
//前面有int,这是定义数组,指定数组包含10个元素

t=a[6];
//这里的a[6]表示引用a数组中序号为6的元素
```

引用一维数组元素

【例6.1】对10个数组元素依次赋值为0,1,2,3,4,5,6,7,8,9，要求按逆序输出。

```
#include<stdio.h>
int main()
{
    int i,a[10];
    for(i=0; i<=9;i++)  //对数组元素a[0]~a[9]赋值
        a[i]=i;
    for(i=9;i>=0;i--)    //输出a[9]~a[0]共10个数组元素
        printf("%d ",a[i]);
    printf("\n");
    return 0;
}
```



第1个for循环使a[0] ~ a[9]的值为0 ~ 9。

a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]	a[9]
0	1	2	3	4	5	6	7	8	9

第2个for循环按a[9]~a[0]的顺序输出各元素的值。

一维数组的初始化

为了使程序简洁，常在定义数组的同时给各数组元素赋值，这称为数组的**初始化**。

(1) 在定义数组时对全部数组元素赋予初值。

```
int a[10]={0,1,2,3,4,5,6,7,8,9};
```

将数组中各元素的初值顺序放在一对花括号内，数据间用逗号分隔。花括号内的数据就称为“**初始化列表**”。

(2) 可以只给数组中的一部分元素赋值。

```
int a[10]={0,1,2,3,4};
```

定义a数组有10个元素，但花括号内只提供5个初值，这表示只给前面5个元素赋初值，系统自动给后5个元素赋初值为0。

(3) 给数组中全部元素赋初值为0。

```
int a[10]={0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
```

或

```
int a[10]={0};      //未赋值的部分元素自动设定为0
```

(4) 在对全部数组元素赋初值时，由于数据的个数已经确定，因此可以不指定数组长度。

```
int a[5]={1,2,3,4,5};
```

或

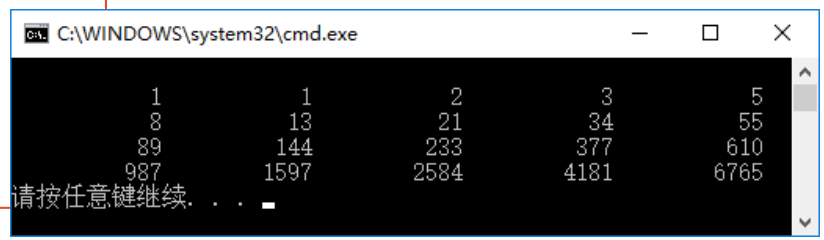
```
int a[ ]={1,2,3,4,5};
```

但是，如果数组长度与提供初值的个数不相同，则方括号中的数组长度不能省略。

一维数组程序举例

【例6.2】用数组来处理求Fibonacci数列问题。

```
#include <stdio.h>
int main()
{
    int i;
    int f[20]={1,1};                    //对最前面两个元素f[0]和f[1]赋初值1
    for(i=2;i<20;i++)
        f[i]=f[i-2]+f[i-1];             //先后求出f[2]~f[19]的值
    for(i=0;i<20;i++)
    {
        if(i%5==0) printf("\n");        //控制每输出5个数后换行
        printf("%12d",f[i]);            //输出一个数
    }
    printf("\n");
    return 0;
}
```

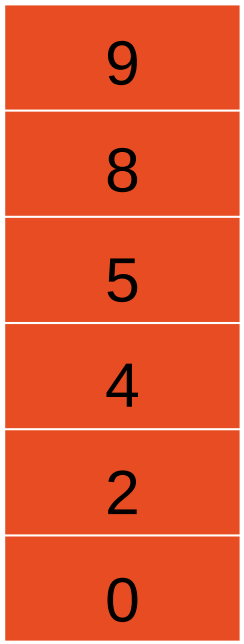


一维数组程序举例

算法

起泡排序法

【例6.3】有10个地区的面积，要求对它们按由小到大的顺序排列。

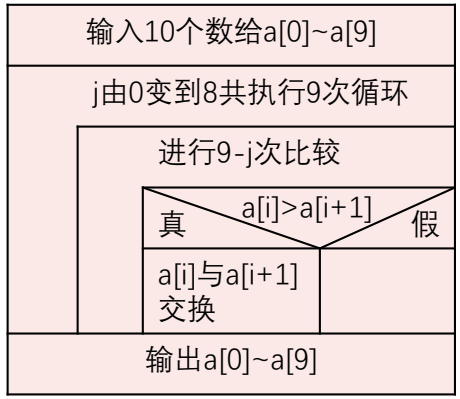


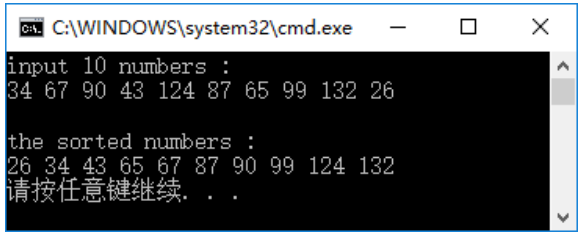
第一趟

一维数组程序举例

【例6.3】有10个地区的面积，要求对它们按由小到大的顺序排列。

```
#include <stdio.h>
int main()
{
    int a[10];
    int i,j,t;
    printf("input 10 numbers :\n");
    for (i=0;i<10;i++)
        scanf("%d",&a[i]);
    printf("\n");
    for(j=0;j<9;j++)                    //进行9次循环，实现9趟比较
        for(i=0;i<9-j;i++)              //在每一趟中进行9-j次比较
            if(a[i]>a[i+1])             //相邻两个数比较
                {t=a[i];a[i]=a[i+1];a[i+1]=t;}
    printf("the sorted numbers :\n");
    for(i=0;i<10;i++)
        printf("%d ",a[i]);
    printf("\n");
    return 0;
}
```





定义和引用二维数组

小例子

有3个小分队，每队有6名队员，要把这些队员的工资用数组保存起来以备查。

	队员1	队员2	队员3	队员4	队员5	队员6
第1分队	2456	1847	1243	1600	2346	2757
第2分队	3045	2018	1725	2020	2458	1436
第3分队	1427	1175	1046	1976	1477	2018

如果建立一个数组pay，它应当是二维的，第一维用来表示第几分队，第二维用来表示第几个队员。例如用$pay_{2,3}$表示2分队第3名队员的工资，它的值是1725。

二维数组常称为**矩阵**(matrix)。把二维数组写成**行**(row)和**列**(column)的排列形式，可以有助于形象化地理解二维数组的逻辑结构。

定义二维数组

类型说明符 数组名[常量表达式][常量表达式]

float pay[3][6];

- float型二维数组
- 数组名为pay
- 数组第二维有6个元素
- 数组第一维有3个元素

```
float a[3][4], b[5][10];
//定义a为3×4(3行4列)的数组，b为5×10(5行10列)的数组
```

```
float a[3, 4], b[5, 10];  //在一对方括号内不能写两个下标
```

二维数组可被看作一种特殊的一维数组: 它的元素又是一个一维数组。

例如，float a[3][4];可以把a看作一个一维数组，它有3个元素：a[0]，a[1]，a[2]，每个元素又是一个包含4个元素的一维数组：

a[0] —— a[0][0] a[0][1] a[0][2] a[0][3]

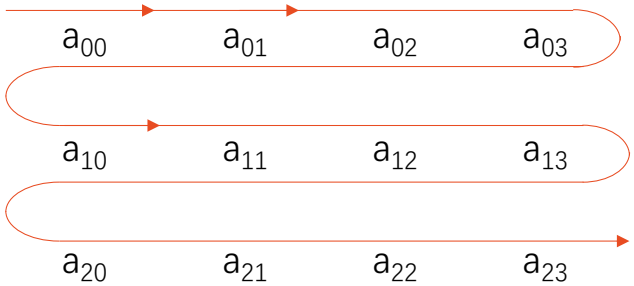
a[1] —— a[1][0] a[1][1] a[1][2] a[1][3]

a[2] —— a[2][0] a[2][1] a[2][2] a[2][3]

二维数组的存储

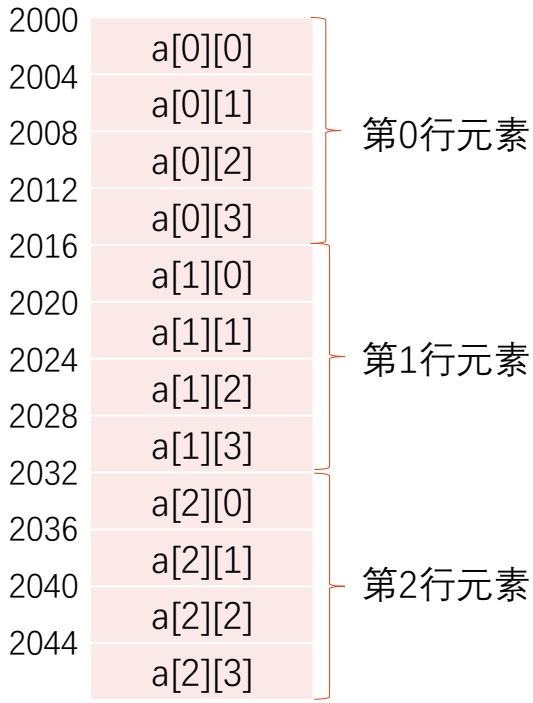
C语言中，二维数组中元素排列的顺序是按行存放的。

float a[3][4]



注意

- 用**矩阵形式**（如3行4列形式）表示二维数组，是**逻辑**上的概念，能形象地表示出行列关系。而在**内存**中，各元素是连续存放的，不是二维的，是**线性**的。



多维数组

```
float a[2, 3, 4];      //定义三维数组a，它有2页，3行，4列
```

多维数组元素在内存中的排列顺序为: 第1维的下标变化最慢，最右边的下标变化最快。

---

float a[2, 3, 4];在内存中的排列顺序为:

a[0][0][0] → a[0][0][1] → a[0][0][2] → a[0][0][3] → a[0][1][0] → a[0][1][1] → a[0][1][2] → a[0][1][3] → a[0][2][0] → a[0][2][1] → a[0][2][2] → a[0][2][3] → a[1][0][0] → a[1][0][1] → a[1][0][2] → a[1][0][3] → a[1][1][0] → a[1][1][1] → a[1][1][2] → a[1][1][3] → a[1][2][0] → a[1][2][1] → a[1][2][2] → a[1][2][3]

引用二维数组元素

数组名[下标] [下标]

“下标”可以是整型常量或整型表达式。

数组元素可以出现在表达式中，也可以被赋值，如：b[1][2]=a[2][3]/2;

注意

- 在引用数组元素时，下标值应在已定义的数组大小的范围内。

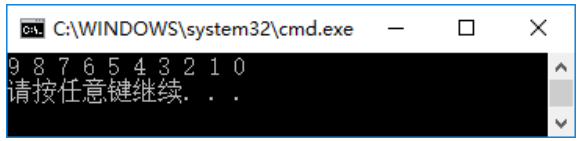
```
int a[3][4];   //定义a为3×4的二维数组

a[3][4]=3;   //不存在a[3][4]元素
//数组a可用的“行下标”的范围为0~2，“列下标”的范围为0~3
```

- 严格区分在**定义**数组时用的a[3][4]和**引用**元素时的a[3][4]的区别。前者用a[3][4]来定义数组的维数和各维的大小，后者a[3][4]中的3和4是数组元素的下标值，a[3][4]代表行序号为3、列序号为4的元素（行序号和列序号均从0起算）。

引用一维数组元素

【例6.1】对10个数组元素依次赋值为0,1,2,3,4,5,6,7,8,9，要求按逆序输出。

```
#include<stdio.h>
int main()
{
    int i,a[10];
    for(i=0; i<=9;i++)  //对数组元素a[0]~a[9]赋值
        a[i]=i;
    for(i=9;i>=0;i--)    //输出a[9]~a[0]共10个数组元素
        printf("%d ",a[i]);
    printf("\n");
    return 0;
}
```



第1个for循环使a[0] ~ a[9]的值为0 ~ 9。

a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]	a[9]
0	1	2	3	4	5	6	7	8	9

第2个for循环按a[9]~a[0]的顺序输出各元素的值。

二维数组的初始化

可以用“初始化列表”对二维数组初始化。

(1)分行给二维数组赋初值。（最清楚直观）

```
int a[3][4]={{1,2,3,4},{5,6,7,8},{9,10,11,12}};
```

(2)可以将所有数据写在一个花括号内，按数组元素在内存中的排列顺序对各元素赋初值。

```
int a[3][4]={1,2,3,4,5,6,7,8,9,10,11,12};
```

(3)可以对部分元素赋初值。

```
int a[3][4]={{1},{5},{9}};          ①
int a[3][4]={{1},{0,6},{0,0,11}};   ②
int a[3][4]={{1},{5,6}};            ③
int a[3][4]={{1},{},{9}};           ④
```

①				②				③				④			
1	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0
5	0	0	0	0	6	0	0	5	6	0	0	0	0	0	0
9	0	0	0	0	0	11	0	0	0	0	0	9	0	0	0

(4)如果对全部元素都赋初值(即提供全部初始数据)，则定义数组时对第1维的长度可以不指定，但第2维的长度不能省。

```
int a[3][4]={1,2,3,4,5,6,7,8,9,10,11,12};   ≡   int a[][4]={1,2,3,4,5,6,7,8,9,10,11,12};
```

在定义时也可以只对部分元素赋初值而省略第1维的长度，但应分行赋初值。

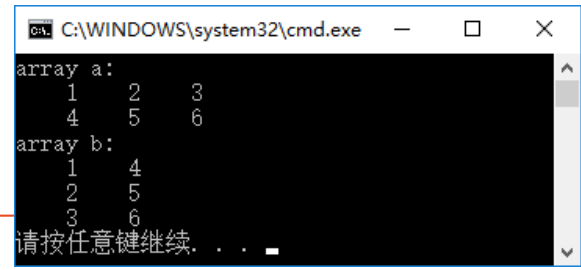
```
int a[][4]={{0,0,3},{},{0,10}};
```

二维数组程序举例

【例6.4】将一个二维数组行和列的元素互换，存到另一个二维数组中。

$$a=\begin{bmatrix}1 & 2 & 3\\ 4 & 5 & 6\end{bmatrix} \Rightarrow b=\begin{bmatrix}1 & 4\\ 2 & 5\\ 3 & 6\end{bmatrix}$$

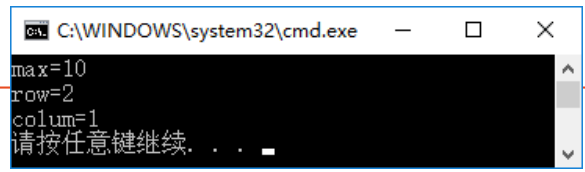
```
#include <stdio.h>
int main()
{
    int a[2][3]={{1,2,3},{4,5,6}};
    int b[3][2],i,j;
    printf("array a:\n");
    for(i=0;i<=1;i++)                  //处理a数组中的一行中各元素
    {
        for (j=0;j<=2;j++)             //处理a数组中某一列中各元素
        {
            printf("%5d",a[i][j]); //输出a数组的一个元素
            b[j][i]=a[i][j];   //将a数组元素的值赋给b数组相应元素
        }
        printf("\n");
    }
    printf("array b:\n");                  //输出b数组各元素
    for(i=0;i<=2;i++)                      //处理b数组中一行中各元素
    {
        for(j=0;j<=1;j++)                  //处理b数组中一列中各元素
            printf("%5d",b[i][j]);         //输出b数组的一个元素
        printf("\n");
    }
    return 0;
}
```



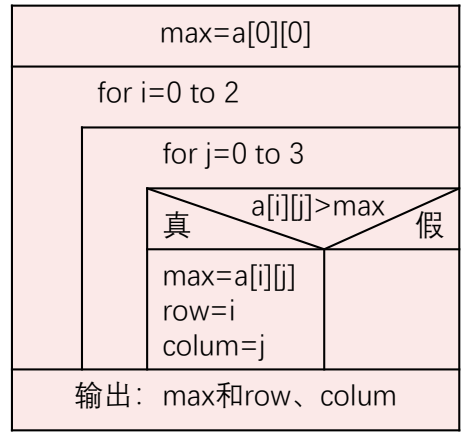
二维数组程序举例

【例6.5】有一个3×4的矩阵，要求编程求出其中值最大的那个元素的值，以及其所在的行号和列号。

```
#include <stdio.h>
int main()
{     int i,j,row=0,colum=0,max;
      int a[3][4]={{1,2,3,4},{9,8,7,6},{-10,10,-5,2}};      //定义数组并赋初值
      max=a[0][0];                                          //先认为a[0][0]最大
      for(i=0;i<=2;i++)
            for(j=0;j<=3;j++)
                  if(a[i][j]>max)                           //如果某元素大于max，就取代max的原值
                  {     max=a[i][j];
                        row=i;                              //记下此元素的行号
                        colum=j;                            //记下此元素的列号
                  }
      printf("max=%d\nrow=%d\ncolum=%d\n",max,row,colum);
      return 0;
}
```



先思考一下在打擂台时怎样确定最后的优胜者。先找出任一人站在台上，第2人上去与之比武，胜者留在台上。再上去第3人，与台上的人(即刚才的得胜者)比武，胜者留台上，败者下台。以后每一个人都是与当时留在台上的人比武。直到所有人都上台比过为止，最后留在台上的就是冠军。



字符数组

定义字符数组

用来存放字符数据的数组是**字符数组**。在字符数组中的一个元素内存放一个字符。

```
char c[10];
c[0]='I'; c[1]=' ';c[2]='a';c[3]='m';c[4]=' ';c[5]='h';c[6]='a';c[7]='p';c[8]='p';c[9]='y';
```

c[0]	c[1]	c[2]	c[3]	c[4]	c[5]	c[6]	c[7]	c[8]	c[9]
I	␣	a	m	␣	h	a	p	p	y

由于字符型数据是以整数形式(ASCII代码)存放的，因此也可以用整型数组来存放字符数据。

```
int c[10];
c[0]='a';      //合法，但浪费存储空间
```

字符数组的初始化

对字符数组初始化，最容易理解的方式是用“初始化列表”，把各个字符依次赋给数组中各元素。

```
char c[10]={'I',' ','a','m',' ','h','a','p','p','y'};          //把10个字符依次赋给c[0]~c[9]这10个元素
```

如果在定义字符数组时不进行初始化，则数组中各元素的值是**不可预料**的。

如果花括号中提供的初值个数（即字符个数）大于数组长度，则出现语法错误。

如果初值个数小于数组长度，则只将这些字符赋给数组中前面那些元素，其余的元素自动定为空字符(即'\0')。

```
char c[10]={'c',' ','p','r','o','g','r','a','m'};
```

c[0]	c[1]	c[2]	c[3]	c[4]	c[5]	c[6]	c[7]	c[8]	c[9]
c	␣	p	r	o	g	r	a	m	\0

如果提供的初值个数与预定的数组长度相同，在定义时可以省略数组长度，系统会自动根据初值个数确定数组长度。

```
char c[]={'I',' ','a','m',' ','h','a','p','p','y'};          //数组c的长度自动定为10
```

也可以定义和初始化一个二维字符数组。

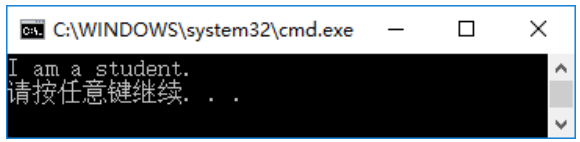
```
char diamond[5][5]={{' ',' ',' ','*'},{' ',' ','*',' ',' ','*'},{'*',' ',' ',' ',' ','*'},{' ',' ','*',' ',' ','*'},{' ',' ',' ','*'}};
```

```
   *
  * *
 *   *
  * *
   *
```

引用字符数组中的元素

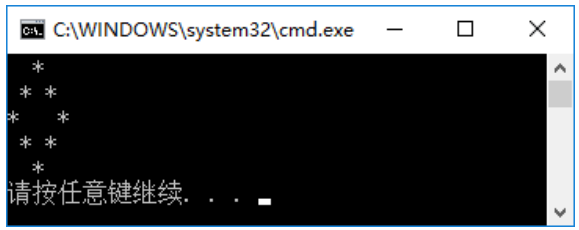
【例6.6】输出一个已知的字符串。

```
#include <stdio.h>
int main()
{
    char c[15]={'I',' ','a','m',' ','a',' ','s','t','u','d','e','n','t','.'};
    int i;
    for(i=0;i<15;i++)
        printf("%c",c[i]);
    printf("\n");
    return 0;
}
```



【例6.7】输出一个菱形图。

```
#include <stdio.h>
int main()
{   char diamond[][5]={{' ',' ','*'},{' ','*',' ','*'},{'*',' ',' ',' ','*'},
    {' ','*',' ','*'},{' ',' ','*'}};
    int i,j;
    for (i=0;i<5;i++)
    {       for (j=0;j<5;j++)
        printf("%c",diamond[i][j]);
        printf("\n");
    }
    return 0;
}
```



字符串和字符串结束标志

在C语言中，是将字符串作为字符数组来处理的。

在实际工作中，人们关心的往往是字符串的有效长度而不是字符数组的长度。

为了测定字符串的实际长度，C语言规定了一个“字符串结束标志”，以字符**'\0'**作为结束标志。

"C program" 字符串是存放在一维数组中的，占10个字节，字符占9个字节，最后一个字节'\0'是由系统自动加上的

注意

- C系统在用字符数组存储字符串常量时会自动加一个'\0'作为结束符。
- 在定义字符数组时应估计实际字符串长度，保证数组长度始终大于字符串实际长度。
- 如果在一个字符数组中先后存放多个不同长度的字符串，则应使数组长度大于最长的字符串的长度。

字符串和字符串结束标志

printf("How do you do?\n");
在向内存中存储时，系统自动在最后一个字符'\n'的后面加了一个'\0'，作为字符串结束标志。在执行printf函数时，每输出一个字符检查一次，看下一个字符是否为'\0'，遇'\0'就停止输出。

char c[]={"I am happy"};
或 char c[]="I am happy";
用一个字符串(注意字符串的两端是用**双引号**而不是单引号括起来的)作为字符数组的初值。

注意

- 数组c的长度不是10，而是11。因为字符串常量的最后由系统加上一个'\0'。

char c[]={'I', ' ', 'a','m', ' ','h','a','p','p','y','\0'};

≠

char c[]={'I', ' ', 'a','m', ' ','h','a','p','p','y'};

char c[10]={"China"};
数组c的前5个元素为: 'C','h','i','n','a',第6个元素为'\0'，后4个元素也自动设定为空字符。

C	h	i	n	a	\0	\0	\0	\0	\0

字符数组的输入输出

(1) 逐个字符输入输出。用格式符“%c”输入或输出一个字符。

(2) 将整个字符串一次输入或输出。用“%s”格式符，意思是对字符串(string)的输入输出。

```
#include <stdio.h>
int main()
{
	char c[15]={'I',' ','a','m',' ','a',' ','s','t','u','d','e','n','t','.'};
	int i;
	for(i=0;i<15;i++)
		printf("%c",c[i]);
	printf("\n");
	return 0;
}
```

```
#include <stdio.h>
int main()
{
	char c[]="China";
	printf("%s\n",c);
	return 0;
}
```

(1) 输出的字符中不包括结束符'\0'。

(2) 用“%s”格式符输出字符串时，printf函数中的输出项是字符数组名，而不是数组元素名。

(3) 如果数组长度大于字符串的实际长度，也只输出到遇'\0'结束。

(4) 如果一个字符数组中包含一个以上'\0'，则遇第一个'\0'时输出就结束。

字符数组的输入输出

```
char c[6];
scanf("%s",c);
```

从键盘输入:

China↙

系统会自动在China后面加一个'\0'结束符。

```
char str1[5],str2[5],str3[5];
scanf("%s%s%s",str1,str2,str3);
```

如果利用一个scanf函数输入多个字符串，则应在输入时以**空格**分隔。

从键盘输入:

How are you? ↙

由于有空格字符分隔，作为3个字符串输入。

str1:	H	o	w	\0	\0
str2:	a	r	e	\0	\0
str3:	y	o	u	?	\0

```
char str[13];
scanf("%s",str);
```

从键盘输入:

How are you? ↙

由于系统把空格字符作为输入的字符串之间的分隔符，因此只将空格前的字符"How"送到str中。

H	o	w	\0	\0	\0	\0	\0	\0	\0	\0	\0	\0

字符数组的输入输出

注意

- scanf函数中的输入项如果是字符数组名，**不要再加地址符&**，因为在C语言中数组名代表该数组第一个元素的地址(或者说数组的起始地址)。

```
scanf("%s", &str);                    //str前面不应加&
```

- 若数组占6个字节。数组名c代表地址2000。可以用下面的输出语句得到数组第一个元素的地址。

```
printf("%o",c);        //用八进制形式输出数组c的起始地址
```

	c数组
2000	C
2001	h
2002	i
2003	n
2004	a
2005	\0

```
printf("%s",c);        //用八进制形式输出数组c的起始地址
```

- 实际上是这样执行的： 按字符数组名c找到其数组第一个元素的地址，然后逐个输出其中的字符，直到遇'\0'为止。

使用字符串处理函数

输出字符串的函数

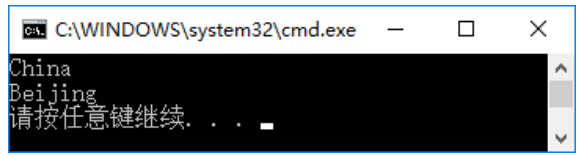
puts(字符数组)

作用：将一个字符串(以'\0'结束的字符序列)输出到终端。

用puts函数输出的字符串中可以包含转义字符。

在用puts输出时将字符串结束标志'\0'转换成'\n'，即输出完字符串后换行。

```
#include <stdio.h>
int main()
{
        char str[]={"China\nBeijing"};
        puts(str);
        return 0;
}
```



输入字符串的函数

gets(字符数组)

作用：从终端输入一个字符串到字符数组，并且得到一个函数值。该函数值是字符数组的起始地址。

```
gets(str);          //str是已定义的字符数组
```

如果从键盘输入:

Computer↙

将输入的字符串"Computer"送给字符数组str（请注意，送给数组的共有9个字符，而不是8个字符），返回的函数值是字符数组str的第一个元素的地址。

注意

- 用puts和gets函数只能输出或输入一个字符串。

```
puts(str1，str2); 或 gets(str1，str2);
```

字符串连接函数

strcat(字符数组1, 字符数组2)

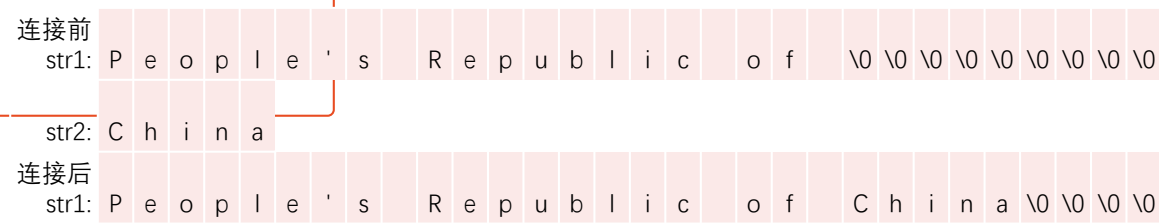
作用：把两个字符数组中的字符串连接起来，把字符串2接到字符串1的后面，结果放在字符数组1中，函数调用后得到一个函数值——字符数组1的地址。

字符数组1必须足够大，以便容纳连接后的新字符串。

连接前两个字符串的后面都有'\0'，连接时将字符串1后面的'\0'取消，只在新串最后保留'\0'。

```
char str1[30]={"People's Republic of "};
char str2[]={"China"};
printf("%s", strcat(str1, str2));
```

输出：People's Republic of China



字符串复制函数

strcpy(字符数组1, 字符串2)

```
char str1[10], str2[]="China";
strcpy(str1, str2); 或 strcpy(str1, "China");
```

执行后，str1：

C	h	i	n	a	\0	\0	\0	\0	\0

作用：将字符串2复制到字符数组1中去。

字符数组1必须定义得足够大，以便容纳被复制的字符串2。字符数组1的长度不应小于字符串2的长度。

“字符数组1”必须写成数组名形式，“字符串2”可以是字符数组名，也可以是一个字符串常量。

若在复制前未对字符数组1初始化或赋值，则其各字节中的内容无法预知，复制时将字符串2和其后的'\0'一起复制到字符数组1中，取代字符数组1中前面的字符，未被取代的字符保持原有内容。

不能用赋值语句将一个字符串常量或字符数组直接给一个字符数组。字符数组名是一个地址常量，它不能改变值，正如数值型数组名不能被赋值一样。

```
str1="China"; str1=str2;
```

可以用strncpy函数将字符串2中前面n个字符复制到字符数组1中去。

```
strncpy(str1, str2, 2);
```

将str2中最前面2个字符复制到str1中，取代str1中原有的最前面2个字符。但复制的字符个数n不应多于str1中原有的字符（不包括'\0'）。

字符串比较函数

strcmp(字符串1, 字符串2)

```
strcmp(str1, str2);
strcmp("China", "Korea");
strcmp(str1, "Beijing");
```

作用：比较字符串1和字符串2。

字符串比较的**规则**是: 将两个字符串自左至右逐个字符相比(按ASCII码值大小比较)，直到出现不同的字符或遇到'\0'为止。

(1) 如全部字符相同，则认为两个字符串相等;

(2) 若出现不相同的字符，则以第1对不相同的字符的比较结果为准。

比较的**结果**由函数值带回。

(1) 如果字符串1与字符串2相同，则函数值为0。

(2) 如果字符串1>字符串2，则函数值为一个正整数。

(3) 如果字符串1<字符串2，则函数值为一个负整数。

注意

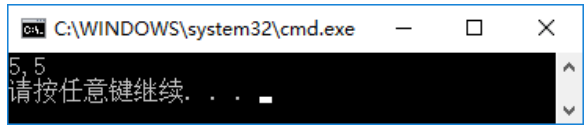
- 对两个字符串比较不能直接用str1>str2进行比较，因为str1和str2代表地址而不代表数组中全部元素，而只能用(strcmp(str1，str2)>0)实现，系统分别找到两个字符数组的第一个元素，然后顺序比较数组中各个元素的值。

测字符串长度的函数

strlen(字符数组)

作用：测试字符串长度的函数。函数的值为字符串中的实际长度(不包括'\0'在内)。

```
#include <stdio.h>
#include <string.h>
int main()
{
        char str[10]="China";
        printf("%d,%d\n",strlen(str),strlen("China"));

}
```



转换为大小写的函数

strlwr(字符串)

作用：将字符串中大写字母换成小写字母。

strupr(字符串)

作用：将字符串中小写字母换成大写字母。

注意

- 以上介绍了常用的8种字符串处理函数，它们属于**库函数**。库函数并非C语言本身的组成部分，而是C语言编译系统为方便用户使用而提供的公共函数。不同的编译系统提供的函数数量和函数名、函数功能都不尽相同，使用时要小心，必要时查一下库函数手册。
- 在使用字符串处理函数时，应当在程序文件的开头用#include <string.h>把string.h文件包含到本文件中。

字符数组应用举例

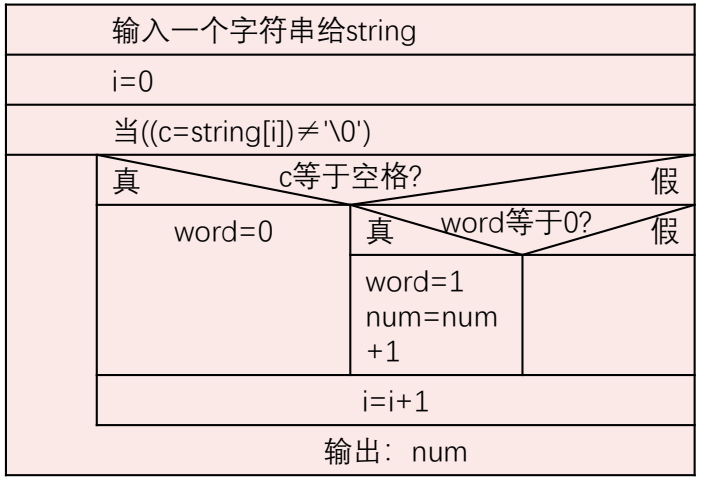
【例6.8】输入一行字符，统计其中有多少个单词，单词之间用空格分隔开。

string：用于存放字符串。

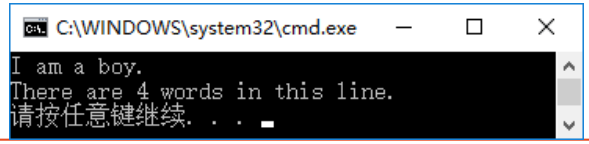
i：计数器，用于遍历字符串中的每个字符。

word：用于判断是否开始了一个新单词的标志。若word=0表示未出现新单词，如出现了新单词，就把word置成1。

num：用于统计单词数。



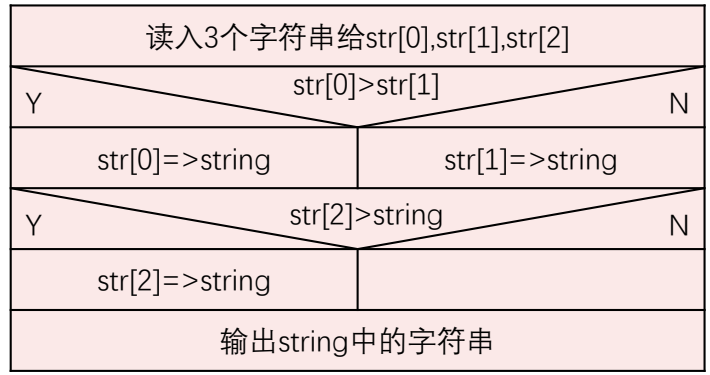

```
#include <stdio.h>
int main()
{
    char string[81];
    int i,num=0,word=0;
    char c;
    gets(string);   //输入一个字符串给字符数组string
    for(i=0;(c=string[i])!='\0';i++)      //只要字符不是'\0'就循环
        if(c==' ') word=0;          //若是空格字符，使word置0
        else if(word==0)    //如果不是空格字符且word原值为0
        {    word=1;        //使word置1
             num++;         //num累加1，表示增加一个单词
        }
    printf("There are %d words in this line.\n",num);   //输出单词数
    return 0;
}
```



字符数组应用举例

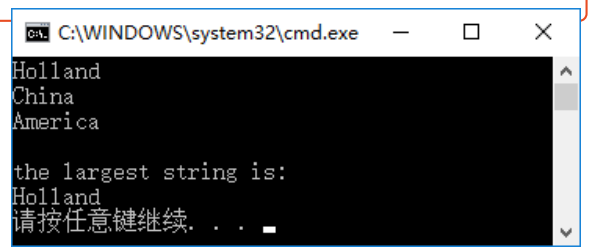
【例6.9】有3个字符串,要求找出其中“最大”者。

str[0]:	H	o	l	l	a	n	d	\0	\0	\0	\0	\0	\0	\0	\0	\0	\0	\0	\0
str[1]:	C	h	i	n	a	\0	\0	\0	\0	\0	\0	\0	\0	\0	\0	\0	\0	\0	\0
str[2]:	A	m	e	r	i	c	a	\0	\0	\0	\0	\0	\0	\0	\0	\0	\0	\0	\0




```
#include<stdio.h>
#include<string.h>
int main()
{
    char str[3][20];      //定义二维字符数组
    char string[20];
    //定义一维字符数组，作为交换字符串时的临时字符数组
    int i;
    for(i=0;i<3;i++)
        gets(str[i]);     //读入3个字符串，分别给str[0],str[1],str[2]
    if(strcmp(str[0],str[1])>0)     //若str[0]大于str[1]
        strcpy(string,str[0]);      //把str[0]的字符串赋给字符数组string
    else                            //若str[0]小于等于str[1]
        strcpy(string,str[1]);      //把str[1]的字符串赋给字符数组string
    if(strcmp(str[2],string)>0)     //若str[2]大于string
        strcpy(string,str[2]);      //把str[2]的字符串赋给字符数组string
    printf("\nthe largest string is:\n%s\n",string);          //输出string
    return 0;
}
```

(1) 流程图和程序注释中的“大于”是指两个字符串的比较中的“大于”。
(2) str[0]，str[1]，str[2]和string是一维字符数组，其中可以存放一个字符串。
(3) strcpy函数在将str[0]，str[1]或str[2]复制到string时，最后都有一个‘\0’。因此，最后用%s格式输出string时，遇到string中第一个‘\0’即结束输出，并不是把string中的全部字符输出。



第 7 章

用函数实现模块化程序设计

为什么要用函数

```
int main()
{
    {
        ⋮    功能1内容
    }

    {
        ⋮    功能2内容
    }

    {
        ⋮    功能1内容
    }

    {
        ⋮    功能2内容
    }
}
```

VS.

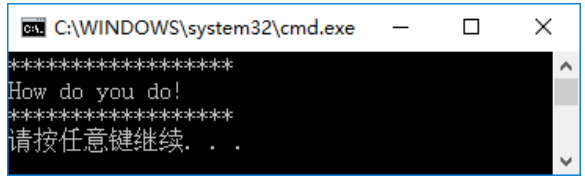
```
功能1函数()
{
    ⋮    功能1内容
}

功能2函数()
{
    ⋮    功能2内容
}

int main()
{
    调用功能1
    调用功能2
    调用功能1
    调用功能2
}
```

- 使用函数可使程序清晰、精炼、简单、灵活。
- 函数就是功能。每一个函数用来实现一个特定的功能。函数名应反映其代表的功能。
- 在设计较大程序时，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。
- 一个C程序可由一个主函数和若干个其他函数构成。由主函数调用其他函数，其他函数也可以互相调用。

为什么要用函数

【例7.1】想输出以下的结果，用函数调用实现。

```
#include <stdio.h>
int main()
{     void print_star();                  //声明print_star函数
      void print_message();               //声明print_message函数
      print_star();                       //调用print_star函数
      print_message();                    //print_message函数
      print_star();                       //调用print_star函数
      return 0;
}

void print_star()                         //定义print_star函数
{     printf("**********\n");//输出一行*号
}

void print_message()                      //定义print_message函数
{     printf("How do you do!\n");  //输出一行文字信息
}
```



print_star和print_message都是用户定义的函数名，分别用来输出一排“*”号和一行文字信息。在定义这两个函数时指定函数的类型为void，意为函数无类型，即无函数值，也就是说，执行这两个函数后不会把任何值带回main函数。

在程序中，定义print_star函数和print_message函数的位置是在main函数的后面，在这种情况下，应当在main函数之前或main函数中的开头部分，对以上两个函数进行“声明”。**函数声明**的作用是把有关函数的信息(函数名、函数类型、函数参数的个数与类型)通知编译系统，以便在编译系统对程序进行编译时，在进行到main函数调用print_star()和print_message()时知道它们是函数而不是变量或其他对象。此外，还对调用函数的正确性进行检查(如类型、函数名、参数个数、参数类型等是否正确)。

为什么要用函数

(1) 一个C程序由一个或多个程序模块组成，每一个程序模块作为一个源程序文件。较大的程序，可分别放在若干个源文件中。这样便于分别编写和编译，提高调试效率。一个源程序文件可以为多个C程序共用。

(2) 一个源程序文件由一个或多个函数以及其他有关内容（如指令、数据声明与定义等）组成。一个源程序文件是一个编译单位，在程序编译时是以源程序文件为单位进行编译的，而不是以函数为单位进行编译的。

(3) C程序的执行是从main函数开始的，如果在main函数中调用其他函数，在调用后流程返回到main函数，在main函数中结束整个程序的运行。

(4) 所有函数都是平行的，即在定义函数时是分别进行的，是互相独立的。一个函数并不从属于另一个函数，即函数不能嵌套定义。函数间可以互相调用，但不能调用main函数。main函数是被操作系统调用的。

(5) 从用户使用的角度看，函数有两种。

① 库函数，它是由系统提供的，用户不必自己定义，可直接使用它们。应该说明，不同的C语言编译系统提供的库函数的数量和功能会有一些不同，当然许多基本的函数是共同的。

② 用户自己定义的函数。它是用以解决用户专门需要的函数。

(6) 从函数的形式看，函数分两类。

① 无参函数。在调用无参函数时，主调函数不向被调用函数传递数据。

② 有参函数。在调用函数时，主调函数在调用被调用函数时，通过参数向被调用函数传递数据。

定义函数

为什么定义函数

C语言要求，在程序中用到的所有函数，必须“**先定义，后使用**”。

定义函数应包括以下几个内容:

(1) 指定函数的名字，以便以后按名调用。

(2) 指定函数的类型，即函数返回值的类型。

(3) 指定函数的参数的名字和类型，以便在调用函数时向它们传递数据。对无参函数不需要这项。

(4) 指定函数应当完成什么操作，也就是函数是做什么的，即函数的功能。这是最重要的，是在函数体中解决的。

定义函数的方法

定义无参函数

```
类型名  函数名()
{
        函数体
}
```

或

```
类型名  函数名(void)
{
        函数体
}
```

函数名后面括号内的void表示“空”，即函数没有参数。

函数体包括**声明部分**和**语句部分**。

在定义函数时要用“类型标识符”(即类型名)指定函数值的类型，即指定函数带回来的值的类型。

定义有参函数

```
类型名  函数名(形式参数表列)
{
        函数体
}
```

```
int max(int x,int y)
{     int z;              //声明部分
      z=x>y?x:y;          //执行语句部分
      return(z);
}
```

定义空函数

```
类型名  函数名()
{}
```

函数体为空，什么也不做。

调用函数

函数调用的形式

函数名(实参表列)

```
print_star();     //调用无参函数
c=max(a,b);       //调用有参函数
```

1. 函数调用语句

把函数调用单独作为一个语句。如printf_star();

这时不要求函数带回值，只要求函数完成一定的操作。

2. 函数表达式

函数调用出现在另一个表达式中，如c=max(a,b);

这时要求函数带回一个确定的值以参加表达式的运算。

3. 函数参数

函数调用作为另一个函数调用时的实参。如m=max(a,max(b,c));， 又如:printf ("%d", max (a,b));

形式参数和实际参数

在调用有参函数时，主调函数和被调用函数之间有数据传递关系。

在定义函数时函数名后面括号中的变量名称为“**形式参数**”（简称“形参”）或“虚拟参数”。

在主调函数中调用一个函数时，函数名后面括号中的参数称为“**实际参数**”（简称“实参”）。 实际参数可以是常量、变量或表达式，但要求它们有确定的值。

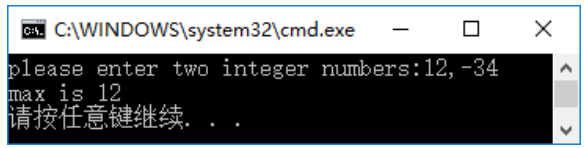
实参与形参的类型应相同或赋值兼容。赋值兼容是指实参与形参类型不同时能按不同类型数值的赋值规则进行转换。

实参和形参间的数据传递

【例7.2】输入两个整数，要求输出其中值较大者。要求用函数来找到大数。

```
#include <stdio.h>
int main()
{    int max(int x,int y);        //对max函数的声明
     int a,b,c;
     printf("please enter two integer numbers:"); //提示输入数据
     scanf("%d,%d",&a,&b);   //输入两个整数
     c=max(a,b);   //调用max函数，有两个实参。大数赋给变量c
     printf("max is %d\n",c);  //输出大数c
     return 0; }
```

```
int max(int x,int y)  //定义max函数，有两个参数
{
     int z;          //定义临时变量z
     z=x>y?x:y;      //把x和y中大者赋给z
     return(z);      //把z作为max函数的值带回main函数
}
```



定义函数，名为max，函数类型为int。指定两个形参x和y，形参的类型为int。

主函数中包含了一个函数调用max(a,b)。max后面括号内的a和b是实参。a和b是在main函数中定义的变量，x和y是函数max的形式参数。通过函数调用，在两个函数之间发生数据传递，实参a和b的值传递给形参x和y，在max函数中把x和y中的大者赋给变量z，z的值作为函数值返回main函数，赋给变量c。

```
c=max(a,b);          (main函数)

int max(int x,int y) (max函数)
{    int z;
     z=x>y?x:y;
     return(z);
}
```

在调用函数过程中发生的实参与形参间的数据传递称为“**虚实结合**”。

函数调用的过程

(1) 在定义函数中指定的形参，在未出现函数调用时，它们并不占内存中的存储单元。在发生函数调用时，函数的形参才被临时分配内存单元。

(2) 将实参的值传递给对应形参。

(3) 在执行函数期间，由于形参已经有值，就可以利用形参进行有关的运算。

(4) 通过return语句将函数值带回到主调函数。应当注意返回值的类型与函数类型一致。如果函数不需要返回值，则不需要return语句。这时函数的类型应定义为void类型。

(5) 调用结束，形参单元被释放。注意: 实参单元仍保留并维持原值，没有改变。如果在执行一个被调用函数时，形参的值发生改变，不会改变主调函数的实参的值。因为实参与形参是两个不同的存储单元。

注意

- 实参向形参的数据传递是“值传递”，单向传递，只能由实参传给形参，而不能由形参传给实参。实参和形参在内存中占有不同的存储单元,实参无法得到形参的值。

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**函数值**(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。**一个函数中可以有一个以上的return语句，执行到哪一个return语句，哪一个return语句就起作用。return语句后面的括号可以不要，如“return z;”与“return(z);”等价。return后面的值可以是一个表达式。

(2) **函数值的类型。**函数值的类型在定义函数时指定。

```
int max (float x，float y)          //函数值为整型
char letter (char c1,char c2)       //函数值为字符型
double min (int x,int y)            //函数值为双精度型
```

(3) **在定义函数时指定的函数类型一般应该和return语句中的表达式类型一致。**

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**。

(4) **对于不带回值的函数，应当用定义函数为“void类型”**（或称“空类型”）。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。此时在函数体中不得出现return语句。

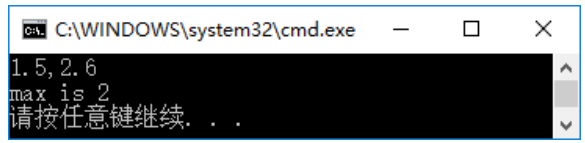
函数的返回值

【例7.3】将例7.2稍作改动，将在max函数中定义的变量z改为float型。

函数返回值的类型与指定的函数类型不同，分析其处理方法。

```
#include <stdio.h>
int main()
{     int max(float x,float y);
      float a,b;
      int c;
      scanf("%f,%f",&a,&b);
      c=max(a,b);
      printf("max is %d\n",c);
      return 0; }
```

```
int max(float x,float y)
{
      float z;          //z为实型变量
      z=x>y?x:y;
      return(z);
}
```



max函数的形参是float型，实参也是float型，在main函数中输入给a和b的值是1.5和2.6。在调用max(a,b)时，把a和b的值1.5和2.6传递给形参x和y。执行函数max中的条件表达式“z=x>y?x:y”，使得变量z得到的值为2.6。现在出现了矛盾: 函数定义为int型，而return语句中的z为float型，要把z的值作为函数的返回值，二者不一致。

怎样处理呢？按赋值规则处理，先将z的值转换为int型，得到2，它就是函数得到的返回值。

如果将main函数中的c改为float型，用%f格式符输出，输出2.000000。因为调用max函数得到的是int型，函数值为整数2。

对被调用函数的声明和函数原型

在一个函数中调用另一个函数（即被调用函数）需要具备如下条件:

(1) 首先被调用的函数必须是已经定义的函数（是库函数或用户自己定义的函数）。

(2) 如果使用库函数，应该在本文件开头用#include指令将调用有关库函数时所需用到的信息“包含”到本文件中来。

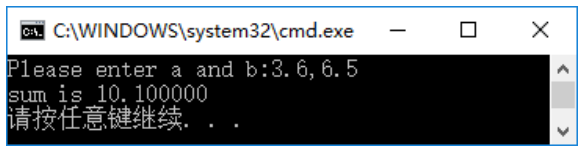
(3) 如果使用用户自己定义的函数，而该函数的位置在调用它的函数（即主调函数）的后面（在同一个文件中），应该在主调函数中对被调用的函数作声明(declaration)。声明的作用是把函数名、函数参数的个数和参数类型等信息通知编译系统，以便在遇到函数调用时，编译系统能正确识别函数并检查调用是否合法。

对被调用函数的声明和函数原型

【例7.4】输入两个实数，用一个函数求出它们之和。

```
#include <stdio.h>
int main()
{    float add(float x, float y);          //对add函数作声明
     float a,b,c;
     printf("Please enter a and b:");      //提示输入
     scanf("%f,%f",&a,&b);                 //输入两个实数
     c=add(a,b);                           //调用add函数
     printf("sum is %f\n",c);              //输出两数之和
     return 0;
}
```

```
float add(float x,float y)        //定义add函数
{    float z;
     z=x+y;
     return(z);                   //把变量z的值作为函数值返回
}
```



函数的声明和函数定义中的第1行（函数首部）基本上是相同的，只差一个分号(函数声明比函数定义中的首行多一个分号)。
函数的首行(即函数首部)称为**函数原型**(function prototype)。
因为在函数的首部包含了检查调用函数是否合法的基本信息(它包括了函数名、函数值类型、参数个数、参数类型和参数顺序)，因此，在函数调用时检查函数原型是否与函数声明一致。这样就能保证函数的正确调用。

对被调用函数的声明和函数原型

函数类型 函数名(参数类型1 参数名1, 参数类型2 参数名2, …, 参数类型n 参数名n);

或 **函数类型 函数名(参数类型1, 参数类型2, …, 参数类型n);**

在函数声明中的形参名可以省写，而只写形参的类型。

```
float add(float x, float y);
float add(float, float);          //不写参数名，只写参数类型
float add(float a, float b);      //参数名不用x,y，而用a,b。合法
```

如果已在文件的开头(在所有函数之前)，已对本文件中所调用的函数进行了声明，则在各函数中不必对其所调用的函数再作声明。

```
char letter(char, char); float f(float,float); int i (float,float);
//所有函数之前，且在函数外部进行函数声明
int main() { … }
//在main函数中要调用letter，f和i函数，不必再对所调用的这3个函数进行声明
char letter(char c1,char c2) { … }        //定义letter函数
float f(float x,float y) { … }            //定义f函数
int i(float j,float k) { … }              //定义i函数
```

注意

- 对函数的“定义”和“声明”不是同一回事。函数的定义是指对函数功能的确立，包括指定函数名、函数值类型、形参及其类型以及函数体等，它是一个完整的、独立的函数单位。而函数的声明的作用则是把函数的名字、函数类型以及形参的类型、个数和顺序通知编译系统，以便在调用该函数时系统按此进行对照检查，它不包含函数体。

函数的嵌套调用

C语言的函数定义是互相平行、独立的，也就是说，在定义函数时，一个函数内不能再定义另一个函数，即不能嵌套定义，但可以嵌套调用函数，即在调用一个函数的过程中，又调用另一个函数。

① 执行main函数的开头部分；

② 遇函数调用语句，调用函数a，流程转去a函数；

③ 执行a函数的开头部分；

④ 遇函数调用语句，调用函数b，流程转去函数b；

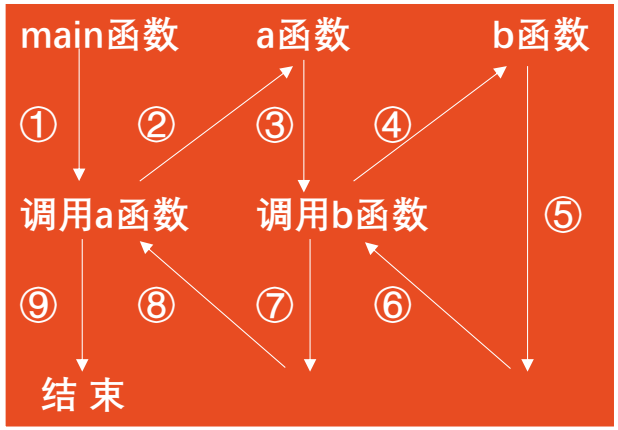
⑤ 执行b函数，如果再无其他嵌套的函数，则完成b函数的全部操作；

⑥ 返回到a函数中调用b函数的位置；

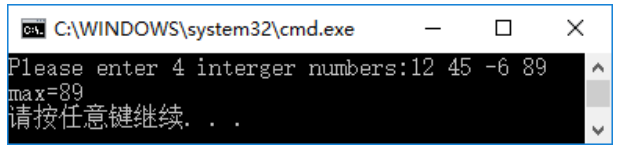
⑦ 继续执行a函数中尚未执行的部分，直到a函数结束；

⑧ 返回main函数中调用a函数的位置；

⑨ 继续执行main函数的剩余部分直到结束。



函数的嵌套调用



【例7.5】输入4个整数，找出其中最大的数。用函数的嵌套调用来处理。

```
#include <stdio.h>
int main()
{    int max4(int a,int b,int c,int d);     //对max4的函数声明
     int a,b,c,d,max;
     printf("Please enter 4 interger numbers:");  //提示输入4个数
     scanf("%d %d %d %d",&a,&b,&c,&d);          //输入4个数
     max=max4(a,b,c,d); //调用max4函数，得到4个数中的最大者
     printf("max=%d \n",max);     //输出4个数中的最大者
     return 0;
}

int max4(int a,int b,int c,int d)      //定义max4函数
{    int max2(int a,int b);           //对max2的函数声明
     int m;
     m=max2(a,b); //调用max2函数，得到a和b中的大者，放在m中
     m=max2(m,c);//调用max2函数，得到a,b,c中的大者，放在m中
     m=max2(m,d);//调用max2函数，得到a,b,c,d中的大者，放在m中
     return(m);      //把m作为函数值带回main函数
}

int max2(int a,int b)      //定义max2函数
{    if(a>=b)
          return a;       //若a≥b，将a作为函数返回值
     else
          return b;       //若a<b，将b作为函数返回值
}
```

在主函数中要调用max4函数，因此在主函数的开头要对max4函数作声明。在max4函数中3次调用max2函数，因此在max4函数的开头要对max2函数作声明。由于在主函数中没有直接调用max2函数，因此在主函数中不必对max2函数作声明，只须在max4函数中作声明即可。max4函数执行过程: 第1次调用max2函数得到的函数值是a和b中的大者，把它赋给变量m，第2次调用max2得到m和c中的大者，也就是a,b,c中的最大者，再把它赋给变量m。第3次调用max2得到m和d中的大者，也就是a,b,c,d中的最大者，再把它赋给变量m。这是一种**递推**方法，先求出2个数的大者；再以此为基础求出3个数的大者；再以此为基础求出4个数的大者。m的值一次一次地变化，直到实现最终要求。

程序改进

(1) 可以将max2函数的函数体改为只用一个return语句，返回一个条件表达式的值:

```
int max2(int a,int b)   //定义max2函数
{return(a>=b?a:b);}   //返回条件表达式的值，即a和b中的大者
```

(2) 在max4函数中，3个调用max2的语句可以用以下一行代替:

```
m=max2(max2(max2(a,b),c),d); //把函数调用作为函数参数
```

甚至可以取消变量m，max4函数可写成

```
int max4(int a,int b,int c,int d)
{    int max2(int a,int b);       //对max2的函数声明
     return max2(max2(max2(a,b),c),d);
}
```

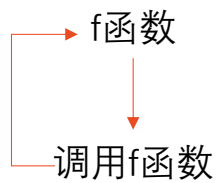
先调用“max2(a,b)”，得到a和b中的大者。再调用“max2(max2(a,b),c)”(其中max2(a,b)为已知)，得到a,b,c三者中的大者。最后由“max2(max2(max2(a,b),c),d)”求得a,b,c,d四者中的大者。

函数的递归调用

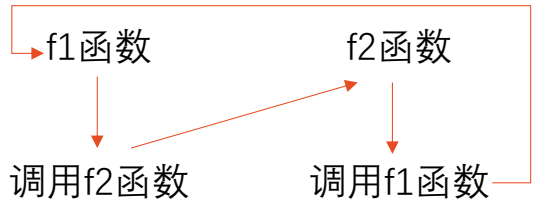
函数的递归调用

在调用一个函数的过程中又出现**直接或间接地调用该函数本身，称为函数的递归调用**。

```
int f(int x)
{
    int y,z;
    z=f(y);   //在执行f函数的过程中又要调用f函数
    return (2*z);
}
```



直接递归



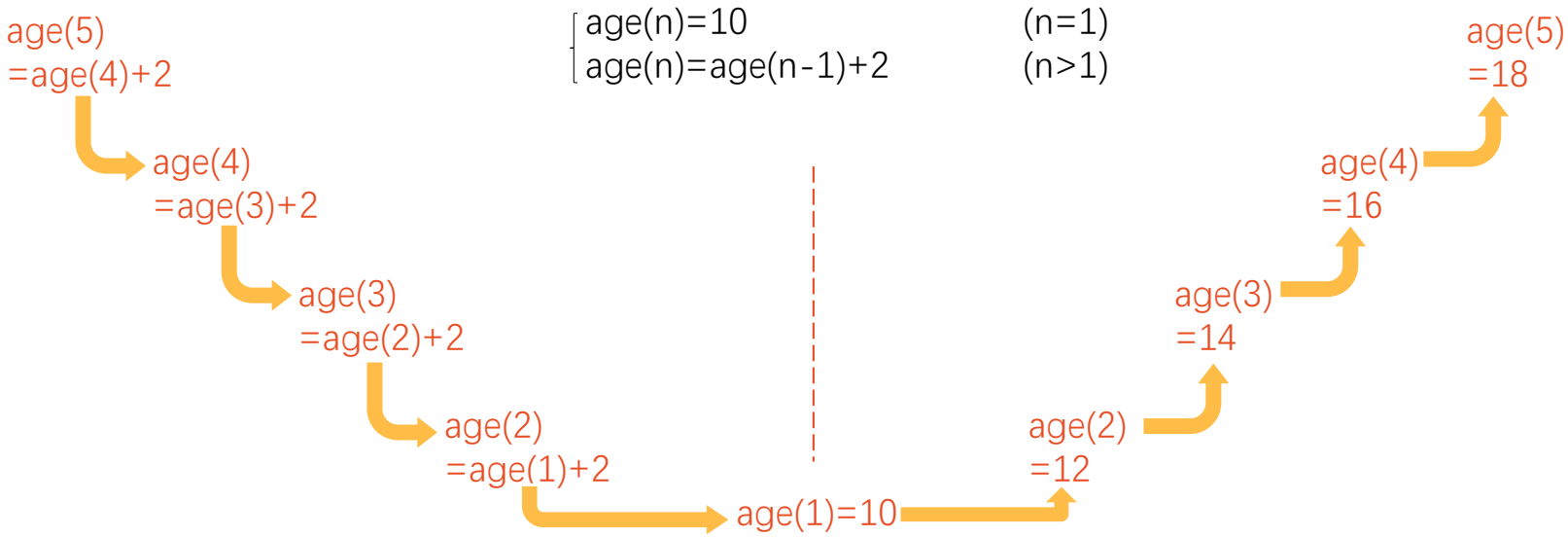
间接递归

程序中不应出现无终止的递归调用，而只应出现有限次数的、有终止的递归调用，这可以用if语句来控制，只有在某一条件成立时才继续执行递归调用；否则就不再继续。

函数的递归调用

【例7.6】有5个学生坐在一起，问第5个学生多少岁，他说比第4个学生大2岁。问第4个学生岁数，他说比第3个学生大2岁。问第3个学生，又说比第2个学生大2岁。问第2个学生，说比第1个学生大2岁。最后问第1个学生，他说是10岁。请问第5个学生多大。

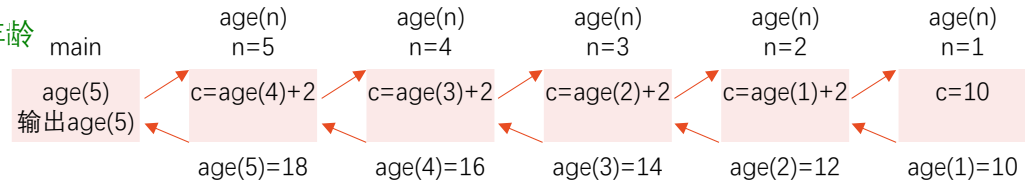
解题思路:



函数的递归调用

【例7.6】有5个学生坐在一起，问第5个学生多少岁，他说比第4个学生大2岁。问第4个学生岁数，他说比第3个学生大2岁。问第3个学生，又说比第2个学生大2岁。问第2个学生，说比第1个学生大2岁。最后问第1个学生，他说是10岁。请问第5个学生多大。

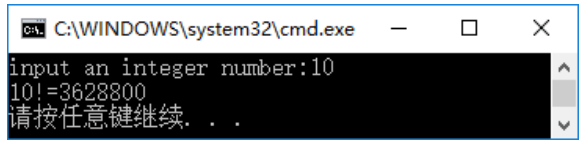
```
#include <stdio.h>
int main()
{     int age(int n);                    //对age函数的声明
      printf("NO.5,age:%d\n",age(5));    //输出第5个学生的年龄
      return 0;
}

int age(int n)                           //定义递归函数
{     int c;                             //c用作存放函数的返回值的变量
      if(n==1)                           //如果n等于1
            c=10;                        //年龄为10
      else                               //如果n不等于1
            c=age(n-1)+2;                //年龄是前一个学生的年龄加2(如第4个学生年龄是第3个学生年龄加2)
      return(c);                         //返回年龄
}
```



注意分析递归的终止条件。

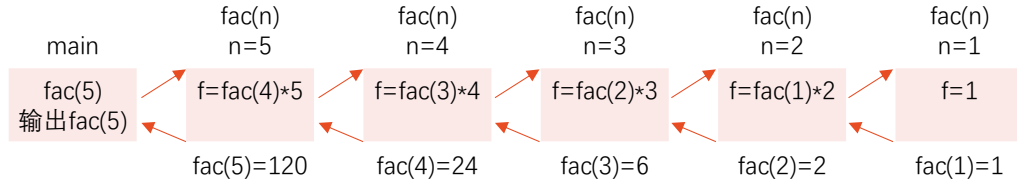
函数的递归调用

【例7.7】用递归方法求n!。



解题思路:

$$n!=\begin{cases} n!=1 & (n=0,1) \\ n\cdot(n-1)! & (n>1) \end{cases}$$




```
#include <stdio.h>
int main()
{     int fac(int n);          //fac函数声明
      int n;
      int y;
      printf("input an integer number:");
      scanf("%d",&n);          //输入要求阶乘的数
      y=fac(n);
      printf("%d!=%d\n",n,y);
      return 0;
}
```

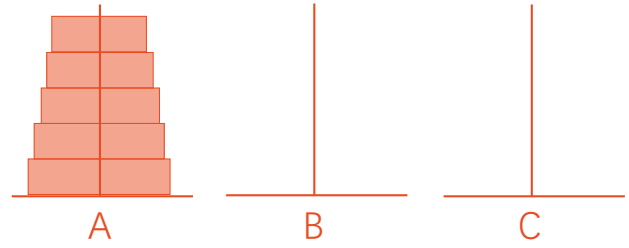
```
int fac(int n)                 //定义fac函数
{
      int f;
      if(n<0)                  //n不能小于0
            printf("n<0,data error!");
      else if(n==0||n==1)      //n=0或,1时n!=1
            f=1;               //递归终止条件
      else
            f=fac(n-1)*n;   //n>1时，n!=n*(n-1)
      return(f);
}
```

注意

- 程序中的变量是int型，如果用Visual C++、GCC以及多数C编译系统为int型数据分配4个字节，能表示的最大数为2 147 483 647，当n=12时，运行正常，输出为479 001 600。如果输入13，企图求13!，是得不到预期结果的，因为求出的结果超过了int型数据的最大值。可将f,y和fac函数定义为float或double型。

函数的递归调用

【例7.8】Hanoi（汉诺）塔问题。古代有一个梵塔，塔内有3个座A,B,C。开始时A座上有64个盘子，盘子大小不等，大的在下，小的在上。有一个老和尚想把这64个盘子从A座移到C座，但规定每次只允许移动一个盘，且在移动过程中在3个座上都始终保持大盘在下，小盘在上。在移动过程中可以利用B座。要求编程序输出移动盘子的步骤。



解题思路:

老和尚会这样想: 假如有另外一个和尚能有办法将上面63个盘子从一个座移到另一座。那么，问题就解决了。此时老和尚只须这样做:

① 命令第2个和尚将63个盘子从A座移到B座;

② 自己将1个盘子（最底下的、最大的盘子）从A座移到C座;

③ 再命令第2个和尚将63个盘子从B座移到C座。

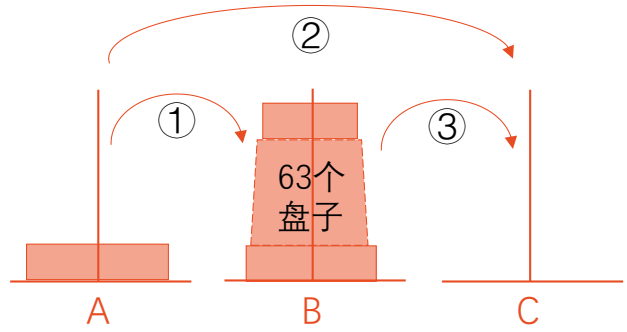
第2个和尚又想: 如果有人能将62个盘子从一个座移到另一座，我就能将63个盘子从A座移到B座，他是这样做的:

① 命令第3个和尚将62个盘子从A座移到C座;

② 自己将1个盘子从A座移到B座;

③ 再命令第3个和尚将62个盘子从C座移到B座。

……



解题思路:

为便于理解，先分析将A座上3个盘子移到C座上的过程:

① 将A座上2个盘子移到B座上（借助C座）。

② 将A座上1个盘子移到C座上。

③ 将B座上2个盘子移到C座上（借助A座）。

其中第②步可以直接实现。第①步又可用递归方法分解为:

将A座上1个盘子从A座移到C座;

将A座上1个盘子从A座移到B座;

将C座上1个盘子从C座移到B座。

第③步可以分解为:

将B座上1个盘子从B座移到A座上;

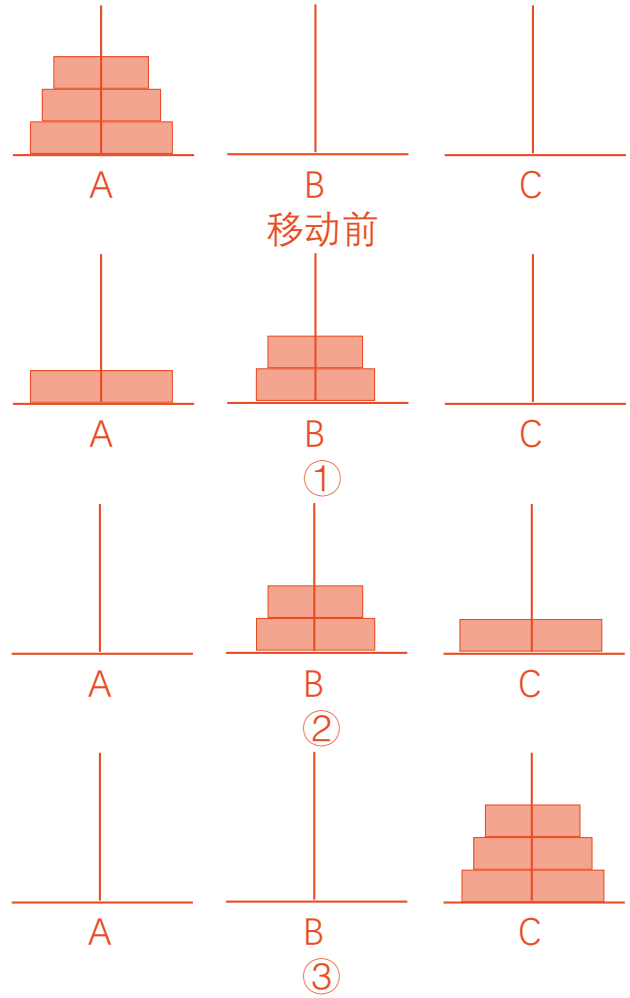
将B座上1个盘子从B座移到C座上;

将A座上1个盘子从A座移到C座上。

将以上综合起来，可得到移动3个盘子的步骤为:

A→C，A→B，C→B，A→C，B→A，B→C，A→C。

共经历7步。由此可推出: 移动n个盘子要经历(2^n-1)步。



解题思路:

由上面的分析可知: 将n个盘子从A座移到C座可以分解为以下3个步骤:

① 将A座上n－1个盘借助C座先移到B座上;

② 把A座上剩下的一个盘移到C座上;

③ 将n－1个盘从B座借助于A座移到C座上。

上面第①步和第③步，都是把n－1个盘从一个座移到另一个座上，采取的办法是一样的，只是座的名字不同而已。为使之一般化，可以将第①步和第③步表示为:

将one座上n－1个盘移到two 座(借助three座)。只是在第①步和第③步中，one,two,three和A,B,C的对应关系不同。对第①步，对应关系是one对应A，two对应B，three对应C。对第③步，是: one对应B，two对应C，three对应A。

因此，可以把上面3个步骤分成两类操作:

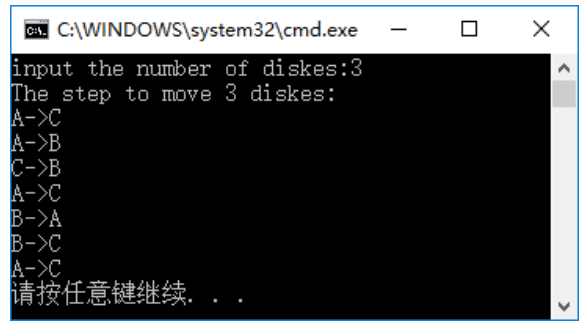
① 将n－1个盘从一个座移到另一个座上（n＞1）。这就是大和尚让小和尚做的工作，它是一个递归的过程，即和尚将任务层层下放，直到第64个和尚为止。——hanoi函数

② 将1个盘子从一个座上移到另一座上。这是大和尚自己做的工作。——move函数

```
#include <stdio.h>
int main()
{
    void hanoi(int n,char one,char two,char three);
    //对hanoi函数的声明
    int m;
    printf("input the number of diskes:");
    scanf("%d",&m);
    printf("The step to move %d diskes:\n",m);
    hanoi(m,'A','B','C');
}
```

```
void hanoi(int n,char one,char two,char three)   //定义hanoi函数
//将n个盘从one座借助two座,移到three座
{   void move(char x,char y);     //对move函数的声明
    if(n==1)
        move(one,three);
    else
    {   hanoi(n-1,one,three,two);
        move(one,three);
        hanoi(n-1,two,one,three);
    }
}
void move(char x,char y)          //定义move函数
{   printf("%c->%c\n",x,y); }
```

在本程序中，调用递归函数hanoi，其终止条件为hanoi函数的参数n的值等于1。显然，此时不必再调用hanoi函数了，直接执行move函数即可。
在本程序中move函数并未真正移动盘子，而只是输出移盘的方案（表示从哪一个座移到哪一个座）。



数组作为函数参数

形式参数	实在参数
变量	常量、变量、表达式、数组元素
数组	数组

数组元素作为函数实参

数组元素可以用作函数实参，但是不能用作形参。因为形参是在函数被调用时临时分配存储单元的，不可能为一个数组元素单独分配存储单元(数组是一个整体，在内存中占连续的一段存储单元)。在用数组元素作函数实参时，把实参的值传给形参，是“**值传递**”方式。数据传递的方向是**从实参传到形参，单向传递**。

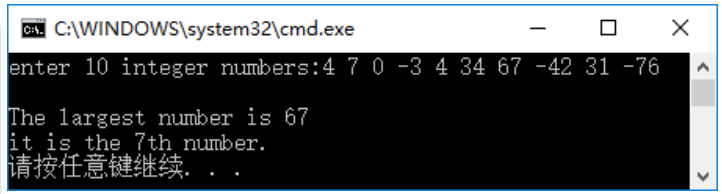
数组元素作函数实参

【例7.9】输入10个数，要求输出其中值最大的元素和该数是第几个数。

```
#include <stdio.h>
int main()
{    int max(int x,int y);              //函数声明
     int a[10],m,n,i;
     printf("enter 10 integer numbers:");
     for(i=0;i<10;i++)                  //输入10个数给a[0]~a[9]
          scanf("%d",&a[i]);
     printf("\n");
     for(i=1,m=a[0],n=0;i<10;i++)
     {    if(max(m,a[i])>m)             //若max函数返回的值大于m
          {    m=max(m,a[i]);          //max函数返回的值取代m原值
               n=i;                    //把此数组元素的序号记下来，放在n中
          }
     }
     printf("The largest number is %d\nit is the %dth number.\n",m,n+1);
}

int max(int x,int y)                    //定义max函数
{    return(x>y?x:y);                   //返回x和y中的大者
}
```

从键盘输入10个数给a[0]~a[9]。变量m用来存放当前已比较过的各数中的最大者。开始时设m的值为a[0]，然后依次将m与a[i]比，如果a[i]大于m，就以a[i]的值取代m的原值。下一次以m的新值与下一个a[i]比较。经过9轮循环的比较，m最后的值就是10个数的最大数。

请注意分析怎样得到最大数是10个数中第几个数。当每次出现以max(m,a[i])的值取代m的原值时，就把i的值保存在变量n中。n最后的值就是最大数的序号(注意序号从0开始)，如果要输出“最大数是10个数中第几个数”，应为n+1。因为数组元素序号从0开始。



一维数组名作函数参数

注意

- 用**数组元素**作实参时，向形参变量传递的是数组元素的**值**，而用**数组名**作函数实参时，向形参(数组名或指针变量) 传递的是**数组首元素的地址**。

【例7.10】有一个一维数组score，内放10个学生成绩，求平均成绩。

```
#include <stdio.h>
int main()
{    float average(float array[10]);      //函数声明
     float score[10],aver;
     int i;
     printf("input 10 scores:\n");
     for(i=0;i<10;i++)
          scanf("%f",&score[i]);
     printf("\n");
     aver=average(score);                 //调用average函数
     printf("average score is %5.2f\n",aver);
     return 0;
}

float average(float array[10])       //定义average函数
{    int i;
     float aver,sum=array[0];
     for(i=1;i<10;i++)
          sum=sum+array[i];          //累加学生成绩
     aver=sum/10;
     return(aver);
}
```

```
C:\WINDOWS\system32\cmd.exe
input 10 scores:
100 56 78 98 67.5 99 54 88.5 76 58

average score is 77.50
请按任意键继续. . .
```

(1) 用数组名作函数参数，应该在主调函数和被调用函数分别定义数组。
(2) 实参数组与形参数组类型必须一致。
(3) 在定义average函数时，声明形参数组的大小为10，但在实际上，指定其大小是不起任何作用的，因为C语言编译系统并不检查形参数组大小，只是将实参数组的首元素的地址传给形参数组名。
(4) 形参数组可以不指定大小，在定义数组时在数组名后面跟一个空的方括号。 float average(float array[])

一维数组名作函数参数

【例7.11】有两个班级，分别有35和30名学生，调用average函数，分别求这两个班的学生的平均成绩。

```
#include <stdio.h>
int main()
{    float average(float array[],int n);
     float score1[5]={98.5,97,91.5,60,55};   //定义长度为5的数组
     float score2[10]={67.5,89.5,99,69.5,77,89.5,76.5,54,60,99.5};
     //定义长度为10的数组
     printf("The average of class A is %6.2f\n",average(score1,5));
     //用数组名score1和5作实参
     printf("The average of class B is %6.2f\n",average(score2,10));
     //用数组名score2和10作实参
     return 0;
}

float average(float array[],int n) //定义average函数，未指定形参数组长度
{    int i;
     float aver,sum=array[0];
     for(i=1;i<n;i++)
          sum=sum+array[i];    //累加n个学生成绩
     aver=sum/n;
     return(aver);
}
```

```
C:\WINDOWS\system32\cmd.exe
The average of class A is  80.40
The average of class B is  78.20
请按任意键继续. . .
```

注意

- **用数组名作函数实参时,不是把数组元素的值传递给形参，而是把实参数组的首元素的地址传递给形参数组，这样两个数组就共占同一段内存单元。**

	a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]	a[9]
起始地址1000	2	4	6	8	10	12	14	16	18	20
	b[0]	b[1]	b[2]	b[3]	b[4]	b[5]	b[6]	b[7]	b[8]	b[9]

一维数组名作函数参数

【例7.12】用选择法对数组中10个整数按由小到大排序。

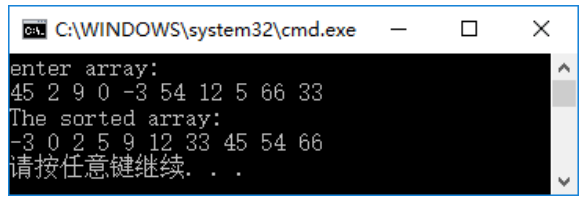
解题思路:

所谓选择法就是先将10个数中最小的数与a[0]对换;再将a[1]~a[9]中最小的数与a[1]对换……每比较一轮,找出一个未经排序的数中最小的一个。共比较9轮。

a[0]	a[1]	a[2]	a[3]	a[4]	
3	6	1	9	4	未排序时的情况
1	6	3	9	4	将5个数中最小的数1与a[0]对换
1	3	6	9	4	将余下的后面4个数中最小的数3与a[1]对换
1	3	4	9	6	将余下的3个数中最小的数4与a[2]对换
1	3	4	6	9	将余下的2个数中最小的数6与a[3]对换，至此完成排序

```
#include <stdio.h>
int main()
{     void sort(int array[],int n);
      int a[10],i;
      printf("enter array:\n");
      for(i=0;i<10;i++)
            scanf("%d",&a[i]);
      sort(a,10);      //调用sort函数,a为数组名，大小为10
      printf("The sorted array:\n");
      for(i=0;i<10;i++)
            printf("%d ",a[i]);
      printf("\n");
      return 0;
}
```

```
void sort(int array[],int n)
{     int i,j,k,t;
      for(i=0;i<n-1;i++)
      {     k=i;
            for(j=i+1;j<n;j++)
                  if(array[j]<array[k])
                        k=j;
            t=array[k]; array[k]=array[i]; array[i]=t;
      }
}
```



可以看到在执行函数调用语句“sort(a,10); ”之前和之后，a数组中各元素的值是不同的。原来是无序的，执行“sort(a,10);”后，a数组已经排好序了，这是由于形参数组array已用选择法进行排序了，形参数组改变也使实参数组随之改变。

多维数组名作函数参数

可以用多维数组名作为函数的实参和形参，在被调用函数中对形参数组定义时可以指定每一维的大小，也可以省略第一维的大小说明。

int array[3][10]; 或 int array[][10];　　//二者等价

int array[][]; 或 int array[3][ ];　　//必须指定列数

在定义二维数组时，必须指定列数(即一行中包含几个元素)，由于形参数组与实参数组类型相同，所以它们是由具有相同长度的一维数组所组成的。不能只指定第1维(行数)而省略第2维(列数)。

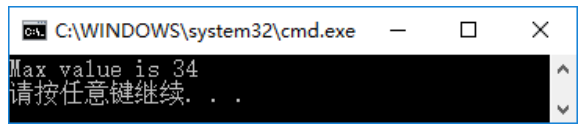
在第2维大小相同的前提下，形参数组的第1维可以与实参数组不同。例如，实参数组定义为int score[5][10];而形参数组定义为int array[ ][10];或int array[8][10];均可以。这时形参数组和实参数组都是由相同类型和大小的一维数组组成的。C语言编译系统不检查第一维的大小。

多维数组名作函数参数

【例7.13】有一个3×4的矩阵，求所有元素中的最大值。

```
#include <stdio.h>
int main()
{     int max_value(int array[][4]);        //函数声明
      int a[3][4]={{1,3,5,7},{2,4,6,8},{15,17,34,12}}; //对数组元素赋初值
      printf("Max value is %d\n",max_value(a));
      //max_value(a)为函数调用
      return 0;
}
```

```
int max_value(int array[][4])  //函数定义
{     int i,j,max;
      max=array[0][0];
      for(i=0;i<3;i++)
            for(j=0;j<4;j++)
                  if(array[i][j]>max) max=array[i][j]; //把大者放在max中
      return(max);
}
```



形参数组array第1维的大小省略，第2维大小不能省略，而且要和实参数组a的第2维的大小相同。在主函数调用max_value函数时，把实参二维数组a的第1行的起始地址传递给形参数组array，因此array数组第1行的起始地址与a数组的第1行的起始地址相同。由于两个数组的列数相同，因此array数组第2行的起始地址与a数组的第2行的起始地址相同。a[i][j]与array[i][j]同占一个存储单元，它们具有同一个值。实际上，array[i][j]就是a[i][j]，在函数中对array[i][j]的操作就是对a[i][j]的操作。

局部变量和全局变量

每一个变量都有一个作用域问题，即它们在什么范围内有效。

局部变量

定义变量可能有3种情况:

(1) 在函数的开头定义;

(2) 在函数内的复合语句内定义;

(3) 在函数的外部定义。

在一个函数内部定义的变量只在本函数范围内有效，也就是说只有在本函数内才能引用它们，在此函数以外是不能使用这些变量的。在复合语句内定义的变量只在本复合语句范围内有效，只有在本复合语句内才能引用它们。在该复合语句以外是不能使用这些变量的，以上这些称为“**局部变量**”。

局部变量

```
float f1(int a)                    //定义函数f1
{int b,c;                          //在函数f1中定义b,c
    ⋮        ├ a，b，c有效
}
char f2(int x,int y)               //定义函数f2
{int i,j;
    ⋮        ├ x，y，i，j有效
}
int main()                         //主函数
{int m,n;
    ⋮        ├ m，n有效
return 0;
}
```

(1) 主函数中定义的变量也只在主函数中有效。主函数也不能使用其他函数中定义的变量。

(2) 不同函数中可以使用同名的变量，它们代表不同的对象，互不干扰。

(3) 形式参数也是局部变量。只在定义它的函数中有效。其他函数中不能直接引用形参。

(4) 在一个函数内部，可以在复合语句中定义变量，这些变量只在本复合语句中有效，这种复合语句也称为“**分程序**”或“**程序块**”。

```
int main ()
{    int a,b;                                          ┐
       ⋮                                               │
     {    int c;                                       │
          c=a+b;  ├ c在此复合语句内有效                 ├ a,b在此范围内有效
            ⋮                                          │
     }                                                 │
       ⋮                                               │
}                                                      ┘
```

全局变量

程序的编译单位是源程序文件,一个源文件可以包含一个或若干个函数。在函数内定义的变量是局部变量,而在函数之外定义的变量称为**外部变量**,外部变量是**全局变量**(也称全程变量)。全局变量可以为本文件中其他函数所共用。它的有效范围为从定义变量的位置开始到本源文件结束。

注意

- **在函数内定义的变量是局部变量，在函数外定义的变量是全局变量。**

全局变量

```
int p=1,q=5;            //定义外部变量
float f1(int a)         //定义函数f1
{
    int b,c;            //定义局部变量
      ⋮
}
char c1,c2;             //定义外部变量
char f2 (int x, int y)  //定义函数f2
{
    int i,j;
      ⋮
}
int main()              //主函数
{
    int m,n;
      ⋮
    return 0;
}
```

全局变量c1,c2的作用范围

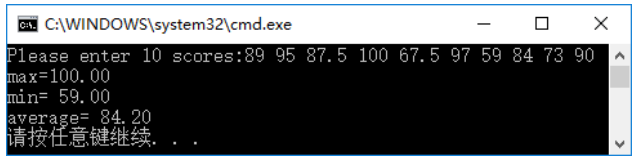
全局变量p,q的作用范围

设置全局变量的作用是增加了函数间数据联系的渠道。由于同一文件中的所有函数都能引用全局变量的值，因此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中全局变量的值。相当于各个函数间有直接的传递通道。由于函数的调用只能带回一个函数返回值，因此有时可以利用全局变量来增加函数间的联系渠道，通过函数调用能得到一个以上的值。

*为了便于区别全局变量和局部变量，在C程序设计人员中有一个习惯（但非规定），将全局变量名的第1个字母用大写表示。

全局变量

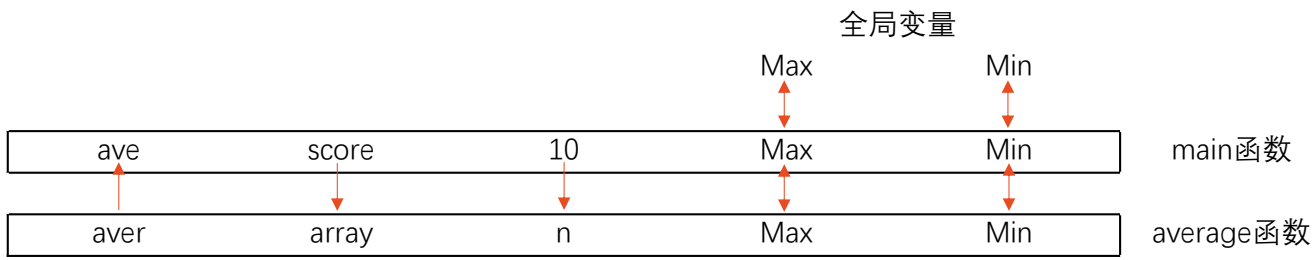
【例7.14】有一个一维数组，内放10个学生成绩，写一个函数，当主函数调用此函数后，能求出平均分、最高分和最低分。




```
#include <stdio.h>
float Max=0,Min=0;      //定义全局变量Max,Min
int main()
{     float average(float array[],int n);
      float ave,score[10];
      int i;
      printf("Please enter 10 scores:");
      for(i=0;i<10;i++)
            scanf("%f",&score[i]);
      ave=average(score,10);
      printf("max=%6.2f\nmin=%6.2f\naverage=%6.2f\n",Max,Min,ave);
      return 0;
}
```

```
float average(float array[],int n)   //定义函数，有一形参是数组
{     int i;
      float aver,sum=array[0];
      Max=Min=array[0];
      for(i=1;i<n;i++)
      {     if(array[i]>Max) Max=array[i];
            else if(array[i]<Min) Min=array[i];
            sum=sum+array[i];
      }
      aver=sum/n;
      return(aver);
}
```

变量的关系：



全局变量

但是，建议不在必要时不要使用全局变量，原因如下:

① 全局变量在程序的全部执行过程中都占用存储单元，而不是仅在需要时才开辟单元。

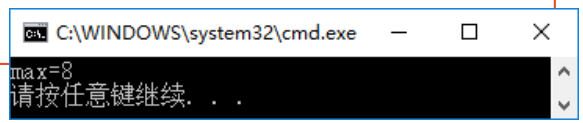
② 它使函数的通用性降低了，因为如果在函数中引用了全局变量，那么执行情况会受到有关的外部变量的影响，如果将一个函数移到另一个文件中，还要考虑把有关的外部变量及其值一起移过去。但是若该外部变量与其他文件的变量同名时，就会出现问题。这就降低了程序的可靠性和通用性。在程序设计中，在划分模块时要求模块的“内聚性”强、与其他模块的“耦合性”弱。即模块的功能要单一（不要把许多互不相干的功能放到一个模块中），与其他模块的相互影响要尽量少，而用全局变量是不符合这个原则的。一般要求把C程序中的函数做成一个相对的封闭体，除了可以通过“实参—形参”的渠道与外界发生联系外，没有其他渠道。这样的程序移植性好，可读性强。

③ 使用全局变量过多，会降低程序的清晰性，人们往往难以清楚地判断出每个瞬时各个外部变量的值。由于在各个函数执行时都可能改变外部变量的值，程序容易出错。因此，要限制使用全局变量。

全局变量

【例7.15】若外部变量与局部变量同名，分析结果。

```
#include <stdio.h>
int a=3,b=5;                    //a,b是全局变量
int main()
{
    int max(int a,int b);//函数声明。a,b是形参
    int a=8;                    //a是局部变量
    printf("max=%d\n",max(a,b));
    return 0;
}

int max(int a,int b)            //a,b是函数形参
{   int c;
    c=a>b?a:b;                  //把a和b中的大者存放在c中
    return(c);
}
```



程序第2行定义了全局变量a和b，并对其初始化。

第3行是main函数，在main函数中(第6行)定义了一个局部变量a。局部变量a的作用范围为第6~8行。在此范围内全局变量a被局部变量a屏蔽，相当于全局变量a在此范围内不存在(即它不起作用)，而全局变量b在此范围内有效。因此第6行中max(a,b)的实参a应是局部变量a，所以max(a,b)相当于max(8,5)。它的值为8。

第10行起定义max函数，形参a和b是局部变量。全局变量a和b在max函数范围内不起作用，所以函数max中的a和b不是全局变量a和b，而是形参a和b，它们的值是由实参传给形参的，即8和5。

变量的存储方式和生存期

动态存储方式与静态存储方式

从变量值**存在的时间**（即**生存期**）来观察，有的变量在程序运行的整个过程都是存在的，而有的变量则是在调用其所在的函数时才临时分配存储单元，而在函数调用结束后该存储单元就马上释放了，变量不存在了。

也就是说，变量的存储有两种不同的方式: **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由系统分配固定的存储空间的方式。

动态存储方式则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。

动态存储方式与静态存储方式

数据分别存放在静态存储区和动态存储区中。全局变量全部存放在静态存储区中，在程序开始执行时给全局变量分配存储区，程序执行完毕就释放。在程序执行过程中它们占据固定的存储单元，而不是动态地进行分配和释放。

在动态存储区中存放以下数据:

① 函数形式参数。在调用函数时给形参分配存储空间。

② 函数中定义的没有用关键字static声明的变量，即自动变量。

③ 函数调用时的现场保护和返回地址等。

对以上这些数据，在函数调用开始时分配动态存储空间，函数结束时释放这些空间。在程序执行过程中，这种分配和释放是动态的，如果在一个程序中两次调用同一函数，而在此函数中定义了局部变量，在两次调用时分配给这些局部变量的存储空间的地址可能是不相同的。

如果一个程序中包含若干个函数，每个函数中的局部变量的生存期并不等于整个程序的执行周期，它只是程序执行周期的一部分。在程序执行过程中，先后调用各个函数，此时会动态地分配和释放存储空间。

存储类别

在C语言中，每一个变量和函数都有两个属性: **数据类型**和**数据的存储类别**。

存储类别指的是数据在内存中存储的方式(如静态存储和动态存储)。

在定义和声明变量和函数时，一般应同时指定其数据类型和存储类别，也可以采用默认方式指定（即如果用户不指定，系统会隐含地指定为某一种存储类别）。

C的存储类别包括4种: **自动的（auto）、静态的（statis）、寄存器的（register）、外部的（extern）**。根据变量的存储类别，可以知道变量的作用域和生存期。

局部变量的存储类别

自动变量(auto变量)

函数中的局部变量，如果不专门声明为static（静态）存储类别，都是动态地分配存储空间的，数据存储在动态存储区中。函数中的形参和在函数中定义的局部变量（包括在复合语句中定义的局部变量），都属于此类。在调用该函数时，系统会给这些变量分配存储空间，在函数调用结束时就自动释放这些存储空间。因此这类局部变量称为**自动变量**。自动变量用关键字auto作存储类别的声明。

```
int f(int a)                    //定义f函数，a为形参
{
      auto int b,c=3;           //定义b,c为自动变量
        ⋮
}
```

实际上，关键字auto可以省略，**不写auto则隐含指定为“自动存储类别”**，它属于动态存储方式。程序中大多数变量属于自动变量。

```
auto int b,c=3;      //等价于int b,c=3;
```

有时希望函数中的局部变量的值在函数调用结束后不消失而继续保留原值，即其占用的存储单元不释放，在下一次再调用该函数时，该变量已有值（就是上一次函数调用结束时的值）。这时就应该指定该局部变量为“**静态局部变量**”，用关键字**static**进行声明。

局部变量的存储类别

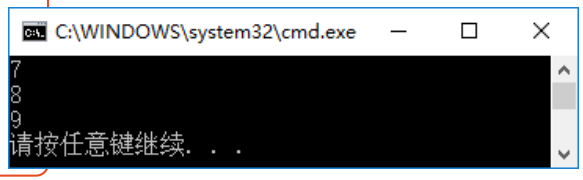
静态局部变量(static局部变量)

【例7.16】考察静态局部变量的值。

```
#include <stdio.h>
int main()
{     int f(int);                    //函数声明
      int a=2,i;                     //自动局部变量
      for(i=0;i<3;i++)
            printf("%d\n",f(a));     //输出f(a)的值
      return 0;
}

int f(int a)
{     auto int b=0;                  //自动局部变量
      static int c=3;                //静态局部变量
      b=b+1;
      c=c+1;
      return(a+b+c);
}
```

静态变量与自动变量的值的比较分析

第几次调用	调用时初值		调用结束时的值		
	b	c	b	c	a+b+c
第1次	0	3	1	4	7
第2次	0	4	1	5	8
第3次	0	5	1	6	9



局部变量的存储类别

静态局部变量(static局部变量)

(1) 静态局部变量属于静态存储类别，在静态存储区内分配存储单元。在程序整个运行期间都不释放。而自动变量（即动态局部变量）属于动态存储类别，分配在动态存储区空间而不在静态存储区空间，函数调用结束后即释放。

(2) 对静态局部变量是在编译时赋初值的，即只赋初值一次，在程序运行时它已有初值。以后每次调用函数时不再重新赋初值而只是保留上次函数调用结束时的值。而对自动变量赋初值，不是在编译时进行的，而是在函数调用时进行的，每调用一次函数重新给一次初值，相当于执行一次赋值语句。

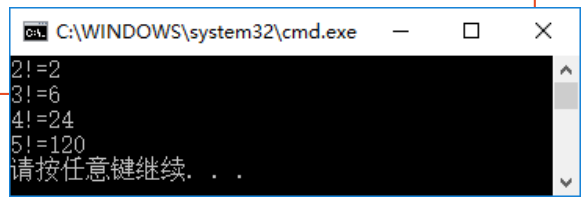
(3) 如果在定义局部变量时不赋初值的话，则对静态局部变量来说，编译时自动赋初值0（对数值型变量）或空字符'\0'（对字符变量）。而对自动变量来说，它的值是一个不确定的值。这是由于每次函数调用结束后存储单元已释放，下次调用时又重新另分配存储单元，而所分配的单元中的内容是不可知的。

(4) 虽然静态局部变量在函数调用结束后仍然存在，但其他函数是不能引用它的。因为它是局部变量，只能被本函数引用，而不能被其他函数引用。

局部变量的存储类别

【例7.17】输出1到5的阶乘值。

```
#include <stdio.h>
int main()
{     int fac(int n);
      int i;
      for(i=1;i<=5;i++)   //先后5次调用fac函数
            printf("%d!=%d\n",i,fac(i));     //每次计算并输出i!的值
      return 0;
}
int fac(int n)
{     static int f=1;          //f保留了上次调用结束时的值
      f=f*n;                   //在上次的f值的基础上再乘以n
      return(f);               //返回值f是n!的值
}
```



(1) 每次调用fac(i)，输出一个i!，同时保留这个i!的值以便下次再乘(i+1)。

(2) 如果函数中的变量只被引用而不改变值，则定义为静态局部变量(同时初始化)比较方便，以免每次调用时重新赋值。

注意

- 用静态存储要多占内存（长期占用不释放，而不能像动态存储那样一个存储单元可以先后为多个变量使用，节约内存），而且降低了程序的可读性，当调用次数多时往往弄不清静态局部变量的当前值是什么。因此，若非必要，不要多用静态局部变量。

局部变量的存储类别

寄存器变量(register变量)

一般情况下，变量（包括静态存储方式和动态存储方式）的值是存放在内存中的。当程序中用到哪一个变量的值时，由控制器发出指令将内存中该变量的值送到运算器中。 经过运算器进行运算，如果需要存数，再从运算器将数据送到内存存放。

如果有一些变量使用频繁（例如，在一个函数中执行10 000次循环，每次循环中都要引用某局部变量），则为存取变量的值要花费不少时间。为提高执行效率，允许将局部变量的值放在CPU中的寄存器中，需要用时直接从寄存器取出参加运算，不必再到内存中去存取。由于对寄存器的存取速度远高于对内存的存取速度，因此这样做可以提高执行效率。这种变量叫做寄存器变量，用关键字register作声明。如

```
register int  f;        //定义f为寄存器变量
```

由于现在的计算机的速度愈来愈快，性能愈来愈高， 优化的编译系统能够识别使用频繁的变量，从而自动地将这些变量放在寄存器中，而不需要程序设计者指定。因此，现在实际上用register声明变量的必要性不大。

注意

- 3种局部变量的存储位置是不同的: 自动变量存储在动态存储区； 静态局部变量存储在静态存储区；寄存器存储在CPU中的寄存器中。

全局变量的存储类别

全局变量都是存放在静态存储区中的。因此它们的生存期是固定的，存在于程序的整个运行过程。

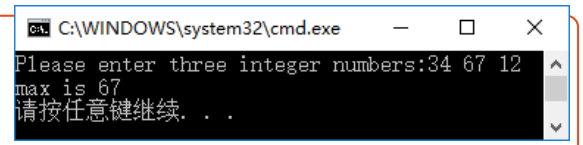
一般来说，外部变量是在函数的外部定义的全局变量，它的作用域是从变量的定义处开始，到本程序文件的末尾。在此作用域内，全局变量可以为程序中各个函数所引用。但有时程序设计人员希望能扩展外部变量的作用域。

全局变量的存储类别

在一个文件内扩展外部变量的作用域

如果外部变量不在文件的开头定义，其有效的作用范围只限于定义处到文件结束。 在定义点之前的函数不能引用该外部变量。如果由于某种考虑，在定义点之前的函数需要引用该外部变量，则应该在引用之前用关键字**extern**对该变量作“**外部变量声明**”，表示把该外部变量的作用域扩展到此位置。有了此声明，就可以从“声明”处起，合法地使用该外部变量。

【例7.18】调用函数，求3个整数中的大者。




```
#include <stdio.h>
int main()
{     int max();
      extern int A,B,C;     //把外部变量A,B,C的作用域扩展到从此处开始
      printf("Please enter three integer numbers:");
      scanf("%d %d %d",&A,&B,&C);       //输入3个整数给A,B,C
      printf("max is %d\n",max());
      return 0;
}
int A,B,C;                  //定义外部变量A,B,C
int max()
{     int m;
      m=A>B?A:B;            //把A和B中的大者放在m中
      if(C>m) m=C;          //将A,B,C三者中的大者放在m中
      return(m);            //返回m的值
}
```

注意

- 提倡将外部变量的定义放在引用它的所有函数之前，这样可以避免在函数中多加一个extern声明。
- 用extern声明外部变量时，类型名可以写也可以省写。例如，“extern int A,B,C;”也可以写成“extern A,B,C;”。因为它不是定义变量，可以不指定类型，只须写出外部变量名即可。

全局变量的存储类别

将外部变量的作用域扩展到其他文件

如果一个程序包含两个文件，在两个文件中都要用到同一个外部变量Num，不能分别在两个文件中各自定义一个外部变量Num，否则在进行程序的连接时会出现“重复定义”的错误。正确的做法是：在任一个文件中定义外部变量Num，而在另一文件中用extern对Num作“外部变量声明”，即“extern Num；”。在编译和连接时，系统会由此知道Num有“外部链接”，可以从别处找到已定义的外部变量Num，并将在另一文件中定义的外部变量Num的作用域扩展到本文件，在本文件中可以合法地引用外部变量Num。

【例7.19】给定b的值，输入a和m，求a*b和a^m的值。

file1.c

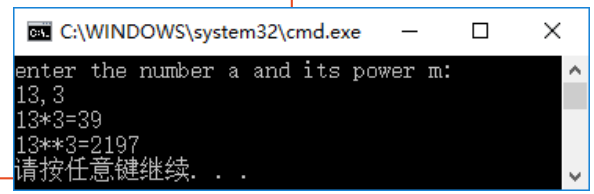
```
#include <stdio.h>
int A;                          //定义外部变量
int main()
{       int power(int);         //函数声明
        int b=3,c,d,m;
        printf("enter the number a and its power m:\n");
        scanf("%d,%d",&A,&m);
        c=A*b;
        printf("%d*%d=%d\n",A,b,c);
        d=power(m);
        printf("%d**%d=%d\n",A,m,d);
        return 0;
}
```

file2.c

```
extern A;
//把file1中定义的外部变量的作用域扩展到本文件
int power(int n)
{       int i,y=1;
        for(i=1;i<=n;i++)
        y*=A;
        return(y);
}
```

注意

- 用这种方法扩展全局变量的作用域应十分慎重，因为在执行一个文件中的操作时，可能会改变该全局变量的值，会影响到另一文件中全局变量的值，从而影响该文件中函数的执行结果。



全局变量的存储类别

将外部变量的作用域扩展到其他文件

extern既可以用来扩展外部变量在本文件中的作用域，又可以使外部变量的作用域从一个文件扩展到程序中的其他文件，系统在编译过程中遇到extern时，

- 先在本文件中找外部变量的定义，如果找到，就在本文件中扩展作用域；
- 如果找不到，就在连接时从其他文件中找外部变量的定义。如果从其他文件中找到了，就将作用域扩展到本文件；
- 如果再找不到，就按出错处理。

全局变量的存储类别

将外部变量的作用域限制在本文件中

有时在程序设计中希望某些外部变量只限于被本文件引用，而不能被其他文件引用。这时可以在定义外部变量时加一个static声明。

file1.c

```
static int A;
int main()
{
        ⋮
}
```

file2.c

```
extern A;
void fun(int n)
{
        ⋮
    A=A*n;        //出错
        ⋮
}
```

这种加上static声明、只能用于本文件的外部变量称为**静态外部变量**。在程序设计中，常由若干人分别完成各个模块，各人可以独立地在其设计的文件中使用相同的外部变量名而互不相干。只须在每个文件中定义外部变量时加上static即可。这就为程序的模块化、通用性提供方便。如果已确认其他文件不需要引用本文件的外部变量，就可以对本文件中的外部变量都加上static，成为静态外部变量，以免被其他文件误用。至于在各文件中在函数内定义的局部变量，本来就不能被函数外引用，更不能被其他文件引用，因此是安全的。

全局变量的存储类别

将外部变量的作用域限制在本文件中

不要误认为对外部变量加static声明后才采取静态存储方式（存放在静态存储区中），而不加static的是采取动态存储（存放在动态存储区）。

声明局部变量的存储类型和声明全局变量的存储类型的含义是不同的。

对于局部变量来说，声明存储类型的作用是指定变量存储的区域(静态存储区或动态存储区)以及由此产生的生存期的问题，而对于全局变量来说，由于都是在编译时分配内存的，都存放在静态存储区，声明存储类型的作用是变量作用域的扩展问题。

全局变量的存储类别

将外部变量的作用域限制在本文件中

用**static**声明一个变量的**作用**是:

(1) 对局部变量用static声明，把它分配在静态存储区，该变量在整个程序执行期间不释放，其所分配的空间始终存在。

(2) 对全局变量用static声明，则该变量的作用域只限于本文件模块(即被声明的文件中)。

注意

- 用auto，register和static声明变量时，是在定义变量的基础上加上这些关键字，而不能单独使用。

```
int a;      //先定义整型变量a
static a;   //企图再将变量a声明为静态变量
```

重新定义

存储类别小结

对一个数据的定义，需要指定两种属性: **数据类型**和**存储类别**，分别使用两个关键字。

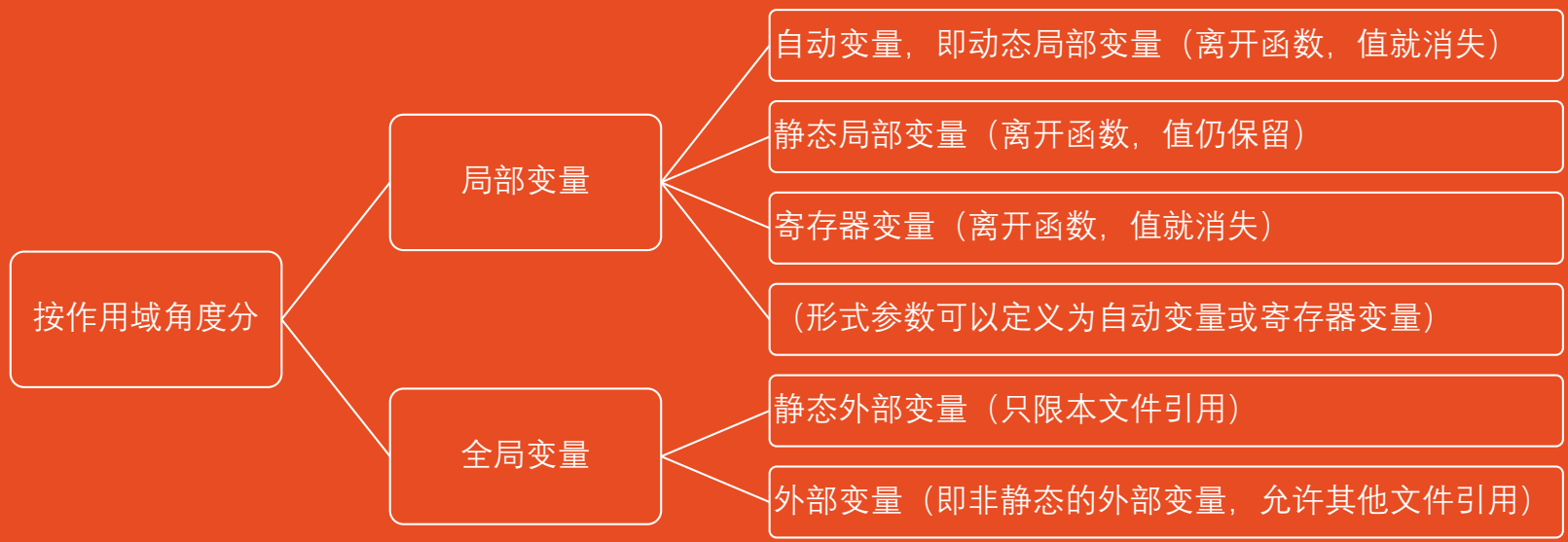
```
static int a;       //静态局部整型变量或静态外部整型变量
auto char c;        //自动变量，在函数内定义
register int d;     //寄存器变量，在函数内定义
```

此外，可以用extern声明已定义的外部变量。

```
extern b;           //将已定义的外部变量b的作用域扩展至此
```

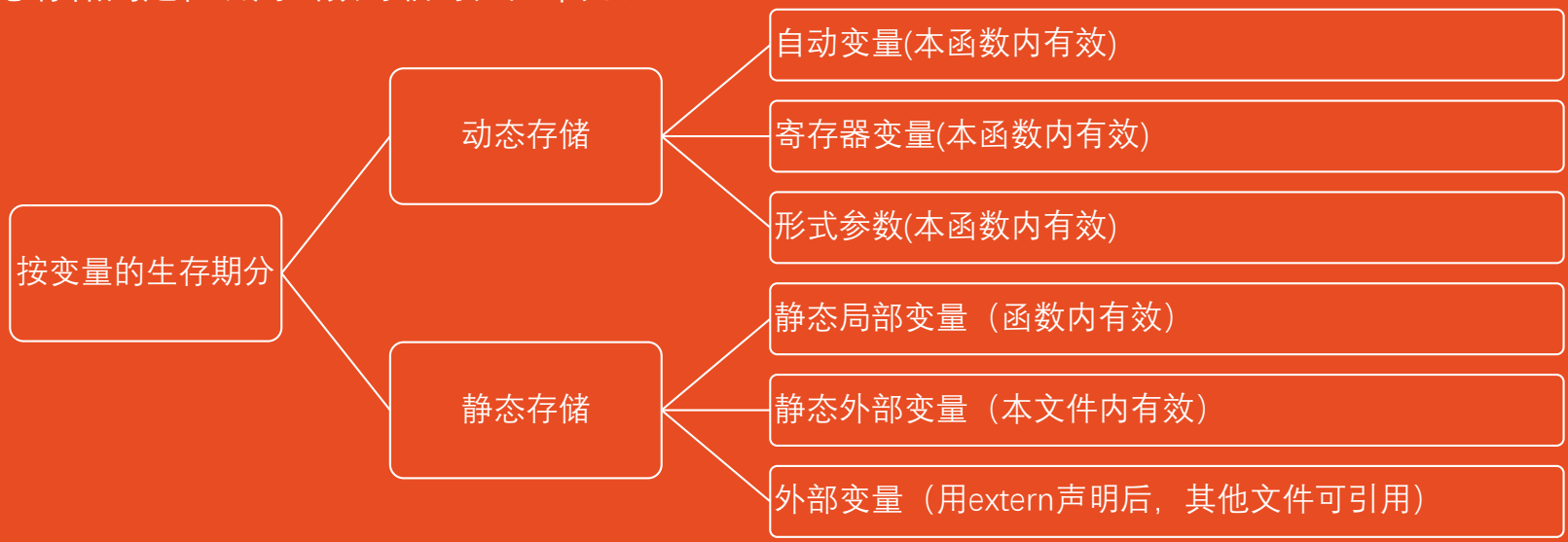
存储类别小结

(1) 从作用域角度分，有局部变量和全局变量。它们采用的存储类别如下:



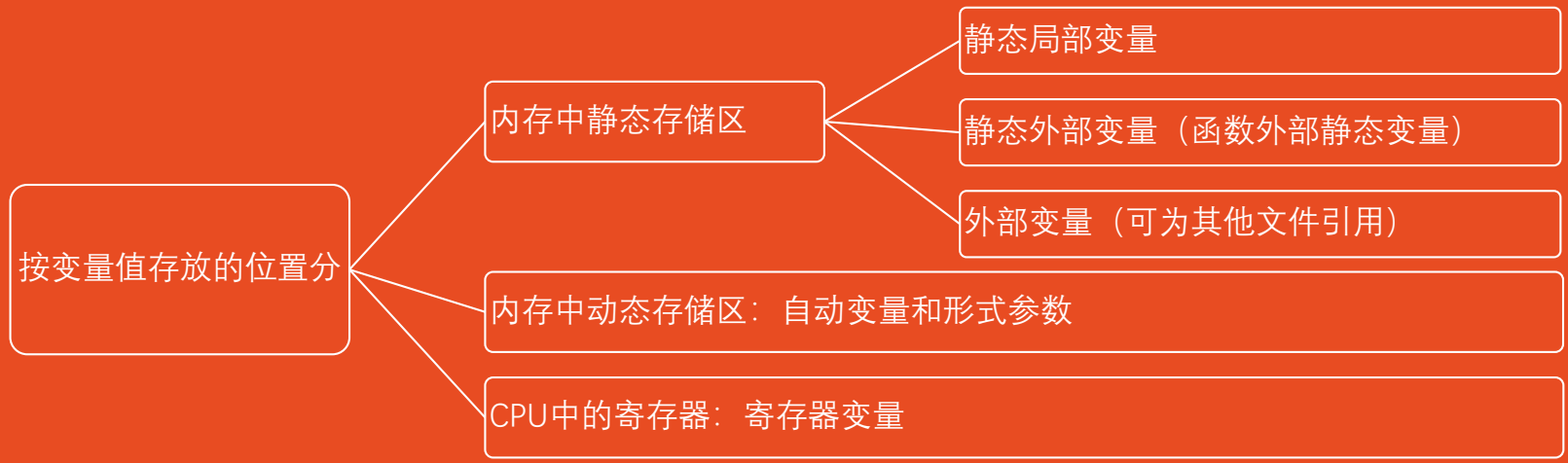
存储类别小结

(2)从变量存在的时间(生存期)来区分,有动态存储和静态存储两种类型。静态存储是程序整个运行时间都存在,而动态存储则是在调用函数时临时分配单元。



存储类别小结

(3)从变量值存放的位置来区分,可分为:



存储类别小结

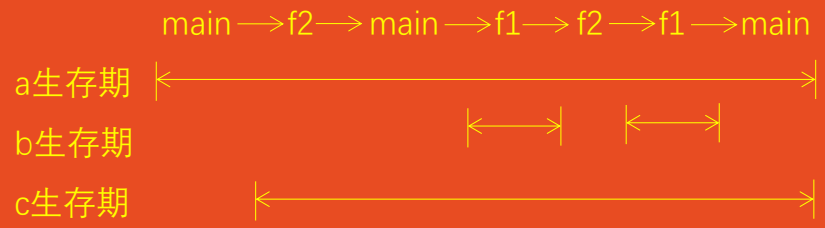
```
int a;
int main()
{
        f2();
        f1();
}
void f1()
{
        auto int b;
        f2();
}
void f2()
{
        static int c;
}
```

a作用域

b作用域

c作用域

(4)关于作用域和生存期的概念。从前面叙述可以知道，对一个变量的属性可以从两个方面分析，一是变量的作用域，一是变量值存在时间的长短，即生存期。前者是从空间的角度，后者是从时间的角度。二者有联系但不是同一回事。



如果一个变量在某个文件或函数范围内是有效的，就称该范围为该变量的**作用域**，在此作用域内可以引用该变量，在专业书中称变量在此作用域内“**可见**”，这种性质称为变量的**可见性**。
如果一个变量值在某一时刻是存在的，则认为这一时刻属于该变量的**生存期**，或称该变量在此时刻“**存在**”。

存储类别小结

各种类型变量的作用域和存在性的情况

变量存储类别	函数内		函数外	
	作用域	存在性	作用域	存在性
自动变量和寄存器变量	√	√	×	×
静态局部变量	√	√	×	√
静态外部变量	√	√	√（只限本文件）	√
外部变量	√	√	√	√

存储类别小结

(5)static对局部变量和全局变量的作用不同。

对局部变量来说，它使变量由动态存储方式改变为静态存储方式。

而对全局变量来说，它使变量局部化(局部于本文件)，但仍为静态存储方式。

从作用域角度看，凡有static声明的，其作用域都是局限的，或者局限于本函数内(静态局部变量)，或者局限于本文件内(静态外部变量)。

关于变量的声明和定义

在声明部分出现的变量有两种情况：一种是需要建立存储空间的(如“int a;”)，另一种是不需要建立存储空间的（如“extern a;”）。前者称为**定义性声明**(defining declaration)，或简称**定义**（definition）；后者称为**引用性声明**(referencing declaration)。一般把**建立存储空间的声明称定义**，而把**不需要建立存储空间的声明称为声明**。

```
int main()
{
        extern A;          //是声明，不是定义。声明将已定义的外部变量A的作用域扩展到此
          ⋮
        return 0;
}
int A;                     //是定义，定义A为整型外部变量
```

外部变量定义和外部变量声明的含义是不同的。外部变量的定义只能有一次，它的位置在所有函数之外。在同一文件中，可以有多次对同一外部变量的声明，它的位置可以在函数之内（哪个函数要用就在哪个函数中声明），也可以在函数之外。系统根据外部变量的定义（而不是根据外部变量的声明）分配存储单元。对外部变量的初始化只能在“定义”时进行，而不能在“声明”中进行。所谓“声明”，其作用是声明该变量是一个已在其他地方已定义的外部变量，仅仅是为了扩展该变量的作用范围而作的“声明”。

注意

- 有一个简单的结论，在函数中出现的对变量的声明(除了用extern声明的以外)都是定义。在函数中对其他函数的声明不是函数的定义。

内部函数和外部函数

函数本质上是全局的，因为定义一个函数的目的就是要被另外的函数调用。如果不加声明的话，一个文件中的函数既可以被本文件中其他函数调用，也可以被其他文件中的函数调用。但是,也可以指定某些函数不能被其他文件调用。根据函数能否被其他源文件调用，将函数区分为**内部函数**和**外部函数**。

内部函数

static 类型名 函数名(形参表);

```
static int fun(int a,int b)
//表示fun是一个内部函数，不能被其他文件调用
```

内部函数又称静态函数，因为它是用**static**声明的。使用内部函数,可以使函数的作用域只局限于所在文件。这样，在不同的文件中即使有同名的内部函数，也互不干扰，不必担心所用函数是否会与其他文件模块中的函数同名。

通常把只能由本文件使用的函数和外部变量放在文件的开头，前面都冠以static使之局部化，其他文件不能引用。这就提高了程序的可靠性。

外部函数

extern 类型名 函数名(形参表);

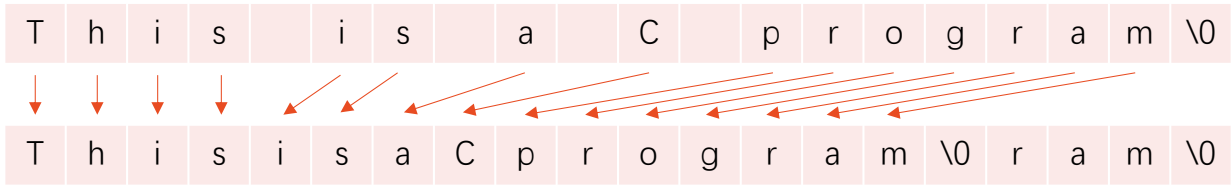
```
extern int fun(int a,int b)
//表示fun可以被其他文件调用
```

在需要调用此函数的其他文件中，需要对此函数作声明(不要忘记，即使在本文件中调用一个函数，也要用函数原型进行声明)。在对此函数作声明时，要加关键字**extern**，表示该函数“是在其他文件中定义的外部函数”。

外部函数

【例7.20】有一个字符串,内有若干个字符,现输入一个字符,要求程序将字符串中该字符删去。用外部函数实现。

解题思路: 设要删除空格



file1.c

```
#include <stdio.h>
int main()
{
    extern void enter_string(char str[]);                //对函数的声明
    extern void delete_string(char str[],char ch);       //对函数的声明
    extern void print_string(char str[]);                //对函数的声明
    //以上3行声明了在本函数中将要调用的已在其他文件中定义的3个函数
    char c,str[80];
    enter_string(str);          //调用在其他文件中定义的enter_string函数
    scanf("%c",&c);             //输入要求删去的字符
    delete_string(str,c);       //调用在其他文件中定义的delete_string函数
    print_string(str);          //调用在其他文件中定义的print_string函数
    return 0;
}
```

file2.c

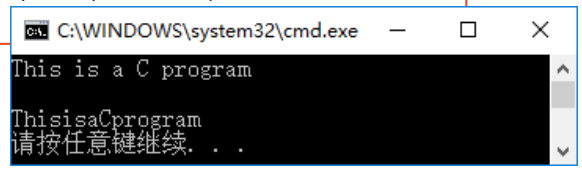
```
void enter_string(char str[80])     //定义外部函数enter_string
{   gets(str);                      //向字符数组输入字符串
}
```

file3.c

```
void delete_string(char str[],char ch)
//定义外部函数delete_string
{   int i,j;
    for(i=j=0;str[i]!='\0';i++)
        if(str[i]!=ch)
            str[j++]=str[i];
    str[j]='\0';
}
```

file4.c

```
void print_string(char str[])
 //定义外部函数print_string
{   printf("%s\n",str);
}
```



外部函数

使用extern声明就能够在本文件中调用在其他文件中定义的函数，或者说把该函数的作用域扩展到本文件。extern声明的形式就是在函数原型基础上加关键字extern。

由于函数在本质上是外部的，在程序中经常要调用其他文件中的外部函数，为方便编程，C语言允许在声明函数时省写extern。

用函数原型能够把函数的作用域扩展到定义该函数的文件之外（不必使用extern）。只要在使用该函数的每一个文件中包含该函数的函数原型即可。函数原型通知编译系统：该函数在本文件中稍后定义，或在另一文件中定义。

利用函数原型扩展函数作用域最常见的例子是#include指令的应用。在#include指令所指定的“头文件”中包含调用库函数时所需的信息。

第 8 章

善 于 利 用 指 针

指 针

如果在程序中定义了一个变量，在对程序进行编译时，系统就会给这个变量分配内存单元。编译系统根据程序中定义的变量类型，分配一定长度的空间。内存区的每一个字节有一个编号，这就是“**地址**”。

由于通过地址能找到所需的变量单元，可以说，**地址指向该变量单元**，将**地址形象化地称为“指针”**。

C语言中的地址包括位置信息(内存编号，或称纯地址)和它所指向的数据的类型信息，或者说它是“带类型的地址”。

存储单元的地址和存储单元的内容是两个不同的概念。

```
int i=1,j=2,k=3;
//设int变量占2字节
```

变量名	地址	内容
i	2000	1
	2001	
j	2002	2
	2003	
k	2004	3
	2005	

在程序中一般是通过变量名来引用变量的值。

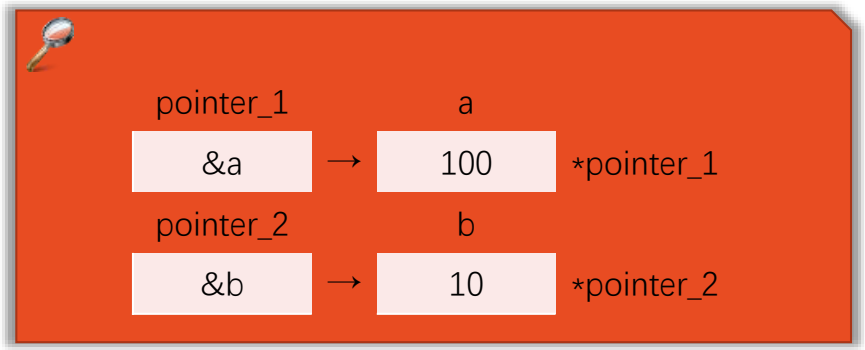
直接按变量名进行的访问，称为“**直接访问**”方式。还可以采用另一种称为“**间接访问**”的方式，即将变量的地址存放在另一变量（**指针变量**）中，然后通过该指针变量来找到对应变量的地址，从而访问变量。

指针变量

使用指针变量的例子

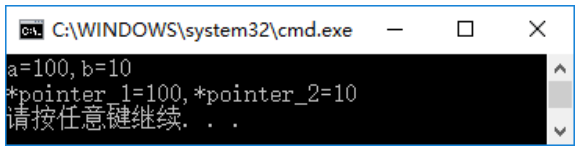
【例8.1】通过指针变量访问整型变量。

```
#include <stdio.h>
int main()
{     int a=100,b=10;
      //定义整型变量a,b，并初始化
      int *pointer_1,*pointer_2;
      //定义指向整型数据的指针变量pointer_1, pointer_2
      pointer_1=&a;      //把变量a的地址赋给指针变量pointer_1
      pointer_2=&b;      //把变量b的地址赋给指针变量pointer_2
      printf("a=%d,b=%d\n",a,b);   //输出变量a和b的值
      printf("*pointer_1=%d,*pointer_2=%d\n",*pointer_1,*pointer_2);
      //输出变量a和b的值
      return 0;
}
```



注意

- 定义指针变量时，左侧应有类型名，否则就不是定义指针变量。



```
*pointer_1;          //企图定义pointer_1为指针变量。出错
```

```
int *pointer_1;      //正确，必须指定指针变量的基类型
```

怎样定义指针变量

类型名 *指针变量名;

```
int *pointer_1, *pointer_2;
```

左端的int是在定义指针变量时必须指定的“**基类型**”。指针变量的基类型用来指定此指针变量可以指向的变量的类型。

前面介绍过基本的数据类型(如int,char，float等)，既然有这些类型的变量，就可以有指向这些类型变量的指针，因此，指针变量是基本数据类型派生出来的类型，它不能离开基本类型而独立存在。

在定义指针变量时要**注意**:

(1) 指针变量前面的“*”表示该变量为指针型变量。指针变量名则不包含“*”。

(2) 在定义指针变量时必须指定基类型。**一个变量的指针的含义包括两个方面，一是以存储单元编号表示的纯地址（如编号为2000的字节），一是它指向的存储单元的数据类型（如int,char,float等）**。

(3) 如何表示指针类型。**指向整型数据的指针类型表示为“int *”，读作“指向int的指针”或简称“int指针”**。

(4) 指针变量中只能存放地址（指针），不要将一个整数赋给一个指针变量。

怎样引用指针变量

① 给指针变量赋值。

② 引用指针变量指向的变量。

③引用指针变量的值。

```
int a, *p;
p=&a;              //把a的地址赋给指针变量p                                          ①
printf("%d",*p);   //以整数形式输出指针变量p所指向的变量的值，即a的值                ②
*p=1;              //将整数1赋给p当前所指向的变量，由于p指向变量a，相当于把1赋给a，即a=1  ②
printf("%o",p);    //以八进制形式输出指针变量p的值，由于p指向a，相当于输出a的地址，即&a   ③
```

注意

- 要熟练掌握两个有关的运算符:

(1) & 取地址运算符。&a是变量a的地址。

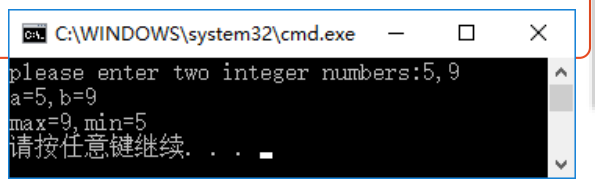
(2) * 指针运算符（或称“间接访问”运算符），*p代表指针变量p指向的对象。

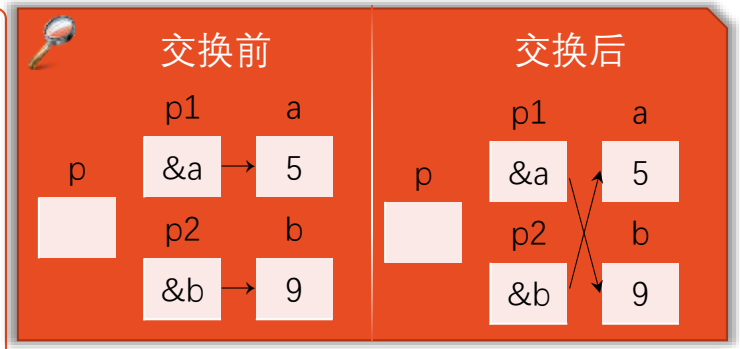
怎样引用指针变量

【例8.2】输入a和b两个整数，按先大后小的顺序输出a和b。

解题思路:不交换整型变量的值，而是交换两个指针变量的值（即a和b的地址）。

```
#include <stdio.h>
int main()
{     int *p1,*p2,*p,a,b;                    //p1,p2的类型是int *类型
      printf("please enter two integer numbers:");
      scanf("%d,%d",&a,&b);                  //输入两个整数
      p1=&a;                                 //使p1指向变量a
      p2=&b;                                 //使p2指向变量b
      if(a<b)                                //如果a<b
      {     p=p1;p1=p2;p2=p;}                //使p1与p2的值互换
      printf("a=%d,b=%d\n",a,b);             //输出a,b
      printf("max=%d,min=%d\n",*p1,*p2);     //输出p1和p2所指向的变量的值
      return 0;
}
```



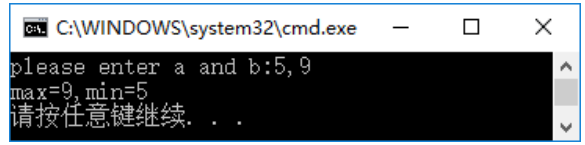


注意

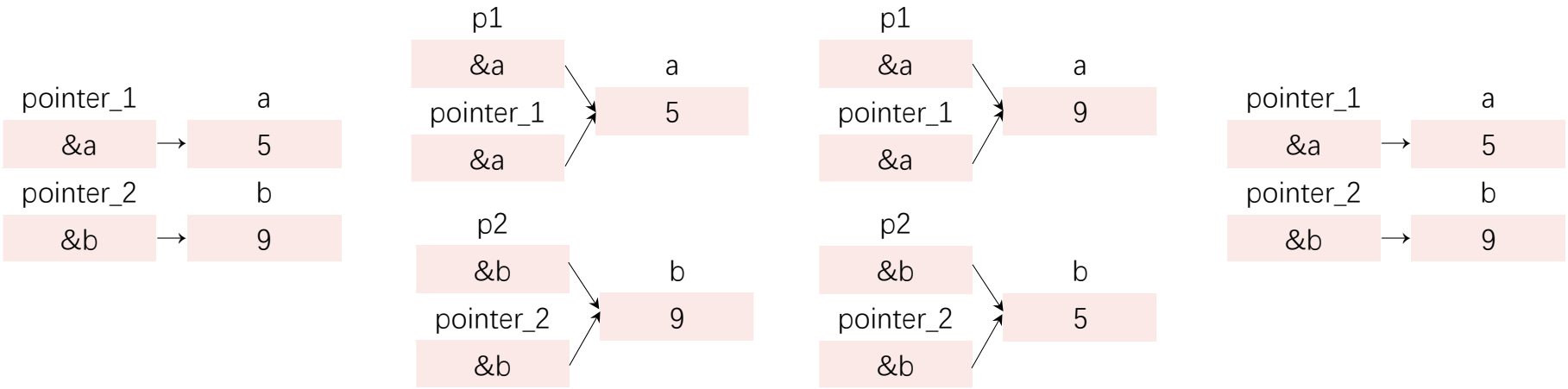
- a和b的值并未交换，它们仍保持原值，但p1和p2的值改变了。
- 实际上，第9行可以改为{p1=&b; p2=&a;}即直接对p1和p2赋以新值，这样可以不必定义中间变量p，使程序更加简练。

指针变量作为函数参数

【例8.3】题目要求同例8.2，即对输入的两个整数按大小顺序输出。现用函数处理，而且用指针类型的数据作函数参数。




```
#include <stdio.h>
int main()
{    void swap(int *p1,int *p2);    //对swap函数的声明
     int a,b;
     int *pointer_1,*pointer_2;     //定义两个int *型的指针变量
     printf("please enter a and b:");
     scanf("%d,%d",&a,&b);          //输入两个整数
     pointer_1=&a;                  //使pointer_1指向a
     pointer_2=&b;                  //使pointer_2指向b
     if(a<b) swap(pointer_1,pointer_2); //如果a<b，调用swap函数
     printf("max=%d,min=%d\n",a,b);   //输出结果
     return 0;
}

void swap(int *p1,int *p2)              //定义swap函数
{    int temp;
     temp=*p1;                          //使*p1和*p2互换
     *p1=*p2;
     *p2=temp;
}    //本例交换a和b的值，而p1和p2的值不变。这恰和例8.2相反
```



指针变量作为函数参数

【例8.3】题目要求同例8.2，即对输入的两个整数按大小顺序输出。现用函数处理，而且用指针类型的数据作函数参数。

```
void swap(int *p1,int *p2)    //定义swap函数
{    int temp;
     temp=*p1;                //使*p1和*p2互换
     *p1=*p2;
     *p2=temp;
}
```

```
void swap(int *p1,int *p2)
{    int *temp;
     *temp=*p1;
     *p1=*p2;
     *p2=*temp;
}
```

*p1就是a，是整型变量。而*temp是指针变量temp所指向的变量。但由于未给temp赋值，因此temp中并无确定的值(它的值是不可预见的)，所以temp所指向的单元也是不可预见的。所以，对*temp赋值就是向一个未知的存储单元赋值，而这个未知的存储单元中可能存储着一个有用的数据，这样就有可能破坏系统的正常工作状况。

```
void swap(int x,int y)
{    int temp;
     temp=x;
     x=y;
     y=temp;
}
```

在函数调用时，a的值传送给x，b的值传送给。执行完swap函数后，x和y的值是互换了，但并未影响到a和b的值。在函数结束时，变量x和y释放了，main函数中的a和b并未互换。

a	b	a	b
5	9	5	9
↓	↓		
5	9	9	5
x	y	x	y

指针变量作为函数参数

函数的调用可以（而且只可以）得到一个返回值（即函数值），而使用指针变量作参数，可以得到多个变化了的值。如果不用指针变量是难以做到这一点的。要善于利用**指针法**。

如果想通过函数调用得到n个要改变的值，可以这样做:

① 在主调函数中设n个变量，用n个指针变量指向它们;

② 设计一个函数，有n个指针形参。在这个函数中改变这n个形参的值;

③ 在主调函数中调用这个函数，在调用时将这n个指针变量作实参，将它们的值，也就是相关变量的地址传给该函数的形参;

④ 在执行该函数的过程中，通过形参指针变量，改变它们所指向的n个变量的值;

⑤ 主调函数中就可以使用这些改变了值的变量。

指针变量作为函数参数

【例8.4】对输入的两个整数按大小顺序输出。

不能企图通过改变指针形参的值而使指针实参的值改变

```
#include <stdio.h>
int main()
{     void swap(int *p1,int *p2);
      int a,b;
      int *pointer_1,*pointer_2;     //pointer_1,pointer_2是int *型变量
      printf("please enter two integer numbers:");
      scanf("%d,%d",&a,&b);
      pointer_1=&a;
      pointer_2=&b;
      if(a<b) swap(pointer_1,pointer_2);
      //调用swap函数，用指针变量作实参
      printf("max=%d,min=%d\n",*pointer_1,*pointer_2);
      return 0;
}

void swap(int *p1,int *p2)    //形参是指针变量
{     int *p;
      p=p1;                   //下面3行交换p1和p2的指向
      p1=p2;
      p2=p;
}
```

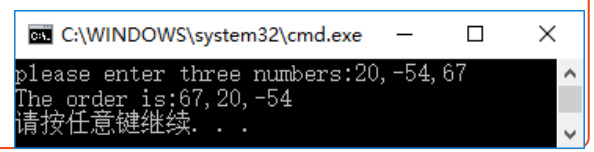
```
C:\WINDOWS\system32\cmd.exe
please enter two integer numbers:5,9
max=5,min=9
请按任意键继续. . .
```

pointer_1	a	p1	a	p1	a	pointer_1	a
&a →	5	&a →	5	&b	5	&a →	5
pointer_2	b	p2	b	p2	b	pointer_2	b
&b →	9	&b →	9	&a	9	&b →	9

指针变量作为函数参数

【例8.5】输入3个整数a,b,c，要求按由大到小的顺序将它们输出。用函数实现。

```
#include <stdio.h>
int main()
{     void exchange(int *q1, int *q2, int *q3);     //函数声明
      int a,b,c,*p1,*p2,*p3;
      printf("please enter three numbers:");
      scanf("%d,%d,%d",&a,&b,&c);
      p1=&a;p2=&b;p3=&c;
      exchange(p1,p2,p3);
      printf("The order is:%d,%d,%d\n",a,b,c);
      return 0;
}

void exchange(int *q1, int *q2, int *q3) //将3个变量的值交换的函数
{     void swap(int *pt1, int *pt2);       //函数声明
      if(*q1<*q2) swap(q1,q2);             //如果a<b，交换a和b的值
      if(*q1<*q3) swap(q1,q3);             //如果a<c，交换a和c的值
      if(*q2<*q3) swap(q2,q3);             //如果b<c，交换b和c的值
}

void swap(int *pt1, int *pt2)        //交换2个变量的值的函数
{     int temp;
      temp=*pt1;                     //交换*pt1和*pt2变量的值
      *pt1=*pt2;
      *pt2=temp;
}
```



通过指针引用数组

数组元素的指针

一个变量有地址，一个数组包含若干元素，每个数组元素都在内存中占用存储单元，它们都有相应的地址。指针变量既然可以指向变量，当然也可以指向数组元素（把某一元素的地址放到一个指针变量中）。**所谓数组元素的指针就是数组元素的地址。**可以用一个指针变量指向一个数组元素。

```
int a[10]={1,3,5,7,9,11,13,15,17,19};      //定义a为包含10个整型数据的数组
int *p;                                    //定义p为指向整型变量的指针变量
p=&a[0];                                   //把a[0]元素的地址赋给指针变量p
```

引用数组元素可以用**下标法**，也可以用**指针法**，即通过指向数组元素的指针找到所需的元素。

```
p=&a[0];    //p的值是a[0]的地址
```

≡

```
p=a;    //p的值是数组a首元素(即a[0])的地址
```

注意

- 程序中的数组名不代表整个数组，只代表数组首元素的地址。

在定义指针变量时可以对它初始化：

```
int *p;
p=&a[0];    //不应写成*p=&a[0];
```

≡

```
int *p=&a[0];
```

≡

```
int *p=a;
```

在引用数组元素时指针的运算

在指针已指向一个数组元素时，可以对指针进行以下运算：

- 加一个整数(用+或+=)，如p+1，表示指向同一数组中的下一个元素；
- 减一个整数(用-或-=)，如p-1，表示指向同一数组中的上一个元素；
- 自加运算，如p++，++p；
- 自减运算，如p--，--p。

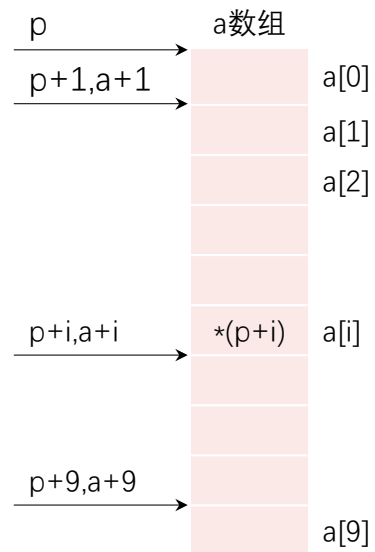
两个指针相减，如p1-p2(只有p1和p2都指向同一数组中的元素时才有意义)，结果是两个地址之差除以数组元素的长度。注意：两个地址不能相加，如p1+p2是无实际意义的。

如果p的初值为&a[0]，则p+i和a+i就是数组元素a[i]的地址，或者说，它们指向a数组序号为i的元素。

(p+i)或(a+i)是p+i或a+i所指向的数组元素，即a[i]。[]实际上是变址运算符，即将a[i]按a+i计算地址，然后找出此地址单元中的值。

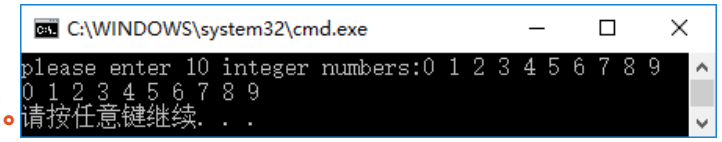
注意

- 执行p+1时并不是将p的值(地址)简单地加1，而是根据定义的基类型加上一个数组元素所占用的字节数。



通过指针引用数组元素

【例8.6】有一个整型数组a，有10个元素，要求输出数组中的全部元素。



①下标法

```
#include <stdio.h>
int main()
{     int a[10];
      int i;
      printf("please enter 10 integer numbers:");
      for(i=0;i<10;i++)
      scanf("%d",&a[i]);
      for(i=0;i<10;i++)
      printf("%d ",a[i]);
      //数组元素用数组名和下标表示
      printf("%\n");
      return 0;
}
```

②通过数组名计算数组元素地址，找出元素的值

```
#include <stdio.h>
int main()
{     int a[10];
      int i;
      printf("please enter 10 integer numbers:");
      for(i=0;i<10;i++)
      scanf("%d",&a[i]);
      for(i=0;i<10;i++)
      printf("%d ",*(a+i));
      //通过数组名和元素序号计算元素地址找到该元素
      printf("\n");
      return 0;
}
```

③用指针变量指向数组元素

```
#include <stdio.h>
int main()
{     int a[10];
      int *p,i;
      printf("please enter 10 integer numbers:");
      for(i=0;i<10;i++)
      scanf("%d",&a[i]);
      for(p=a;p<(a+10);p++)
      printf("%d ",*p);
      //用指针指向当前的数组元素
      printf("\n");
      return 0;
}
```

第(1)和第(2)种方法执行效率是相同的。C编译系统是将a[i]转换为*(a+i)处理的，即先计算元素地址。因此用第(1)和第(2)种方法找数组元素费时较多。

第(3)种方法比第(1)、第(2)种方法快，用指针变量直接指向元素，不必每次都重新计算地址，像p++这样的自加操作是比较快的。这种有规律地改变地址值(p++)能大大提高执行效率。

通过指针引用数组元素

用下标法比较直观，能直接知道是第几个元素。适合初学者使用。

用地址法或指针变量的方法不直观，难以很快地判断出当前处理的是哪一个元素。单用指针变量的方法进行控制，可使程序简洁、高效。

注意

- 在使用指针变量指向数组元素时，有以下几个问题要注意:

(1) 可以通过改变指针变量的值指向不同的元素。

如果不用p变化的方法而用数组名a变化的方法（例如，用a++）行不行呢？

```
for(p=a;a<(p+10);a++)
printf("%d",*a);
```

因为数组名a代表数组首元素的地址，它是一个指针型常量，它的值在程序运行期间是固定不变的。既然a是常量，所以a++是无法实现的。

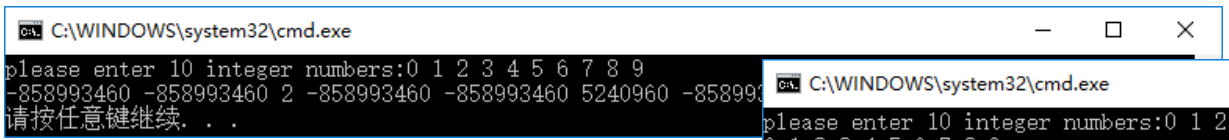
(2) 要注意指针变量的当前值。

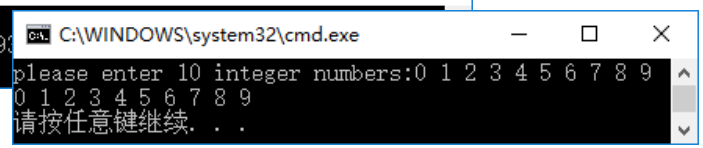
通过指针引用数组元素

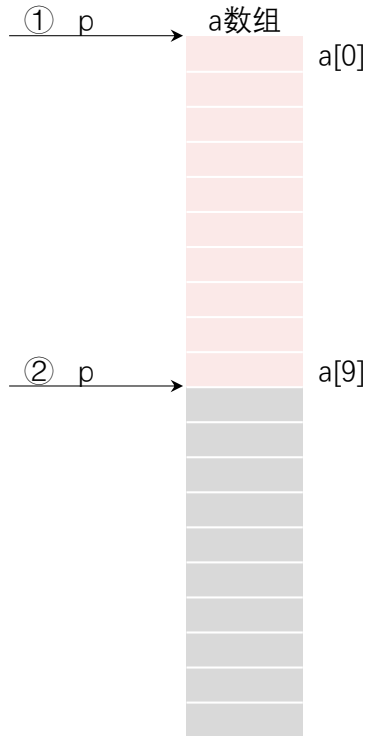
【例8.7】通过指针变量输出整型数组a的10个元素。

```
#include <stdio.h>
int main()
{    int *p,i,a[10];
     p=a;                    //p指向a[0]          ①
     printf("please enter 10 integer numbers:");
     for(i=0;i<10;i++)
     scanf("%d",p++);    //输入10个整数给a[0]~a[9]
     for(i=0;i<10;i++,p++)
     printf("%d ",*p);      //想输出a[0]~a[9] ②
     printf("\n");
     return 0;
}
```

```
#include <stdio.h>
int main()
{    int i,a[10],*p=a;     //p的初值是a，p指向a[0]
     printf("please enter 10 integer numbers:");
     for(i=0;i<10;i++)
     scanf("%d",p++);
     p=a;                    //重新使p指向a[0]
     for(i=0;i<10;i++,p++)
     printf("%d ",*p);
     printf("\n");
     return 0;
}
```







通过指针引用数组元素

(1) 从例8.7可以看到，虽然定义数组时指定它包含10个元素，并用指针变量p指向某一数组元素，但是实际上指针变量p可以指向数组以后的存储单元，结果不可预期，应避免出现这样的情况。

(2) 指向数组元素的指针变量也可以带下标，如p[i]。p[i]被处理成*(p+i)，如果p是指向一个整型数组元素a[0]，则p[i]代表a[i]。但是必须弄清楚p的当前值是什么？如果当前p指向a[3]，则p[2]并不代表a[2]，而是a[3+2]，即a[5]。

(3) 利用指针引用数组元素，比较方便灵活，有不少技巧。请分析下面几种情况：

设p开始时指向数组a的首元素（即p=a）：

①
```
p++;        //使p指向下一元素a[1]
*p;         //得到下一个元素a[1]的值
```

②
```
*p++;    /*由于++和*同优先级，结合方向自右而左，因此它等价于*(p++)。先引用p的值，实现*p的运算，然后再使p自增1*/
```

③
```
*(p++);     //先取*p值，然后使p加1
*(++p);     //先使p加1，再取*p
```

④
```
++(*p);    /*表示p所指向的元素值加1，如果p=a，则相当于++a[0]，若a[0]的值为3，则a[0]的值为4。注意: 是元素a[0]的值加1，而不是指针p的值加1*/
```

⑤ 如果p当前指向a数组中第i个元素a[i]，则：

```
*(p--)      //相当于a[i--]，先对p进行“*”运算，再使p自减
*(++p)      //相当于a[++i]，先使p自加，再进行“*”运算
*(--p)      //相当于a[--i]，先使p自减，再进行“*”运算
```

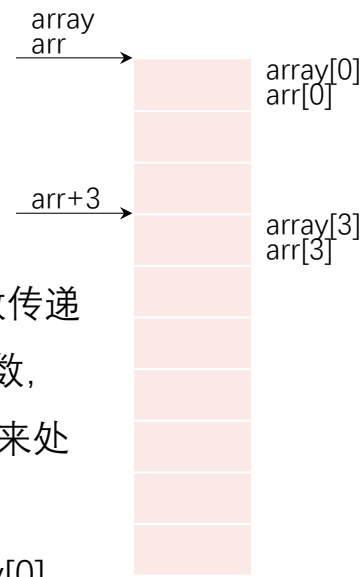
用数组名作函数参数

```
int main()
{     void fun(int arr[], int n); //对fun函数的声明
      int array[10];              //定义array数组
        ⋮
      fun(array,10);              //用数组名作函数的参数
      return 0;
}
void fun(int arr[], int n)        //定义fun函数
{
        ⋮
}
```

≡

```
void fun(int *arr, int n)         //定义fun函数
{
        ⋮
}
```

array是实参数组名，arr为形参数组名。当用数组名作参数时，如果形参数组中各元素的值发生变化，实参数组元素的值随之变化。



在该函数被调用时，系统会在fun函数中建立一个指针变量arr，用来存放从主调函数传递过来的实参数组首元素的地址。如果在fun函数中用运算符sizeof测定arr所占的字节数，可以发现sizeof(arr)的值为4(用Visual C++时)。这就证明了系统是把arr作为指针变量来处理的(指针变量在Visual C++中占4个字节)。

当arr接收了实参数组的首元素地址后，arr就指向实参数组首元素，也就是指向array[0]。

用数组名作函数参数

以变量名和数组名作为函数参数的比较

实参类型	变量名	数组名
要求形参的类型	变量名	数组名或指针变量
传递的信息	变量的值	实参数组首元素的地址
通过函数调用能否改变实参的值	不能改变实参变量的值	能改变实参数组的值

C语言调用函数时虚实结合的方法都是采用“值传递”方式，当用变量名作为函数参数时传递的是变量的值，当用数组名作为函数参数时，由于数组名代表的是数组首元素地址，因此传递的值是地址，所以要求形参为指针变量。

注意

- 实参数组名代表一个固定的地址，或者说是指针常量，但形参数组名并不是一个固定的地址，而是按指针变量处理。

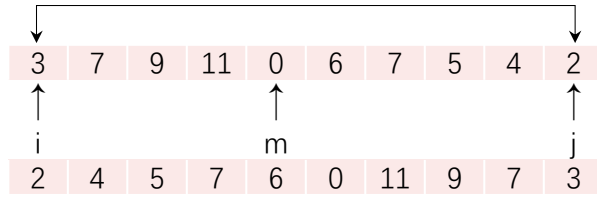
在函数调用进行虚实结合后，形参的值就是实参数组首元素的地址。

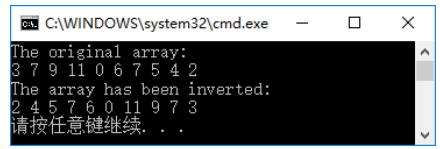
在函数执行期间，它可以再被赋值。

```
void fun (arr[ ],int n)
{    printf("%d\n", *arr);      //输出array[0]的值
     arr=arr+3;                 //形参数组名可以被赋值
     printf("%d\n", *arr);      //输出array[3]的值
}
```

用数组名作函数参数

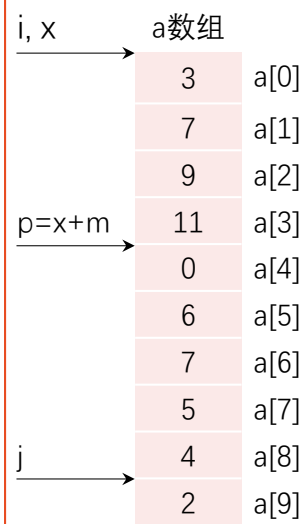
【例8.8】将数组a中n个整数按相反顺序存放。






```c
#include <stdio.h>
int main()
{    void inv(int x[],int n);     //inv函数声明
     int i,a[10]={3,7,9,11,0,6,7,5,4,2};
     printf("The original array:\n");
     for(i=0;i<10;i++)
          printf("%d ",a[i]);     //输出未交换时数组各元素的值
     printf("\n");
     inv(a,10);                   //调用inv函数，进行交换
     printf("The array has been inverted:\n");
     for(i=0;i<10;i++)
          printf("%d ",a[i]);     //输出交换后数组各元素的值
     printf("\n");
     return 0;
}
void inv(int x[],int n)           //形参x是数组名
{    int temp,i,j,m=(n-1)/2;
     for(i=0;i<=m;i++)
     {    j=n-1-i;
          temp=x[i]; x[i]=x[j]; x[j]=temp;     //把x[i]和x[j]交换
     }
     return;
}
```

```c
#include <stdio.h>
int main()
{    void inv(int *x,int n);
     int i,a[10]={3,7,9,11,0,6,7,5,4,2};
     printf("The original array:\n");
     for(i=0;i<10;i++)
          printf("%d ",a[i]);
     printf("\n");
     inv(a,10);
     printf("The array has been inverted:\n");
     for(i=0;i<10;i++)
          printf("%d ",a[i]);
     printf("\n");
     return 0;
}

void inv(int *x,int n)                    //形参x是指针变量
{    int *p,temp,*i,*j,m=(n-1)/2;
     i=x; j=x+n-1; p=x+m;
     for(;i<=p;i++,j--)
     {    temp=*i; *i=*j; *j=temp;}      //*i与*j交换
     return;
}
```



用数组名作函数参数

如果有一个实参数组，要想在函数中改变此数组中的元素的值，实参与形参的对应关系有以下4种情况。

① 形参和实参都用数组名

② 实参用数组名，形参用指针变量。

③ 实参形参都用指针变量。

④ 实参为指针变量，形参为数组名。

①

```
int main()
{     int a[10];
        ⋮
      f(a,10);
        ⋮
}

int f(int x[], int n)
{
        ⋮
}
```

②

```
int main()
{     int a[10];
        ⋮
      f(a,10);
        ⋮
}

int f(int *x, int n)
{
        ⋮
}
```

③

```
int main()
{     int a[10];*p=a;
        ⋮
      f(p,10);
        ⋮
}

int f(int *x, int n)
{
        ⋮
}
```

④

```
int main()
{     int a[10];*p=a;
        ⋮
      f(p,10);
        ⋮
}

int f(int x[], int n)
{
        ⋮
}
```

用数组名作函数参数 【例8.9】改写例8.8，用指针变量作实参。

```
#include <stdio.h>
int main()
{     void inv(int *x,int n);      //inv函数声明
      int i,arr[10],*p=arr;        //指针变量p指向arr[0]
      printf("The original array:\n");
      for(i=0;i<10;i++,p++)
            scanf("%d",p);         //输入arr数组的元素
      printf("\n");
      p=arr;                       //指针变量p重新指向arr[0]
      inv(p,10);                   //调用inv函数，实参p是指针变量
      printf("The array has been inverted:\n");
      for(p=arr;p<arr+10;p++)
            printf("%d ",*p);
      printf("\n");
      return 0;
}

void inv(int *x,int n)             //定义inv函数，形参x是指针变量
{     int *p,m,temp,*i,*j;
      m=(n-1)/2;
      i=x;j=x+n-1;p=x+m;
      for(;i<=p;i++,j--)
      {     temp=*i;*i=*j;*j=temp;}
      return;
}
```

```
#include <stdio.h>
int main()
{     void inv(int *x,int n);      //inv函数声明
      int i,*arr;                  //指针变量arr未指向数组元素
      printf("The original array:\n");
      for(i=0;i<10;i++)
            scanf("%d",arr+i);
      printf("\n");
      inv(arr,10);     //调用inv函数，实参arr是指针变量，但无指向
      printf("The array has been inverted:\n");
      for(i=0;i<10;i++)
            printf("%d ",*(arr+i));
      printf("\n");
      return 0;
}
```

注意

- 如果用指针变量作实参，必须先使指针变量有确定值，指向一个已定义的对象。

用数组名作函数参数

【例8.10】用指针方法对10个整数按由大到小顺序排序。（选择排序法）

```
#include <stdio.h>
int main()
{     void sort(int x[],int n);    //sort函数声明
      int i,*p,a[10];
      p=a;                         //指针变量p指向a[0]
      printf("please enter 10 integer numbers:");
      for(i=0;i<10;i++)
            scanf("%d",p++);       //输入10个整数
      p=a;//指针变量p重新指向a[0]
      sort(p,10);                  //调用sort函数
      for(p=a,i=0;i<10;i++)
      {     printf("%d ",*p);      //输出排序后的10个数组元素
            p++;
      }
      printf("\n");
      return 0;
}
```

```
void sort(int x[],int n)                  //定义sort函数，x是形参数组名
{     int i,j,k,t;
      for(i=0;i<n-1;i++)
      {     k=i;
            for(j=i+1;j<n;j++)
                  if(x[j]>x[k]) k=j;
            if(k!=i)
            {     t=x[i]; x[i]=x[k]; x[k]=t;}
      }
}
```

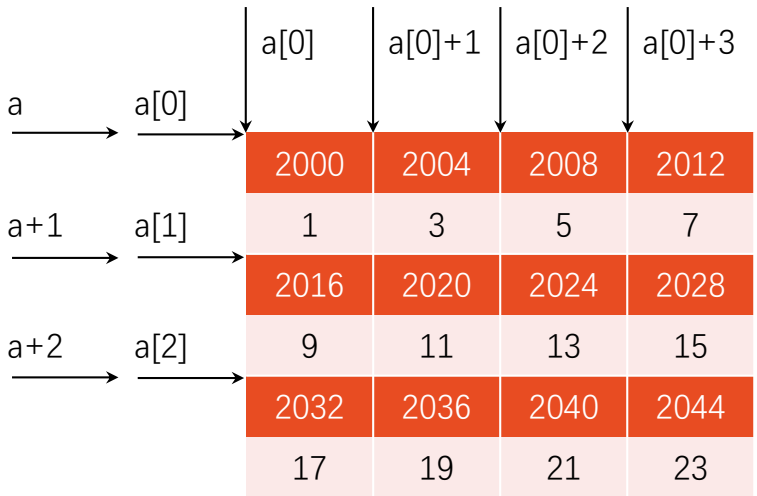
```
void sort(int *x,int n)                   //形参x是指针变量
{     int i,j,k,t;
      for(i=0;i<n-1;i++)
      {     k=i;
            for(j=i+1;j<n;j++)
                  if(*(x+j)>*(x+k)) k=j;      //*(x+j)就是x[j]，其他亦然
            if(k!=i)
            {     t=*(x+i); *(x+i)=*(x+k); *(x+k)=t;}
      }
}
```

```
C:\WINDOWS\system32\cmd.exe
please enter 10 integer numbers:12 34 5 689 -43 56 -21 0 24 65
689 65 56 34 24 12 5 0 -21 -43
请按任意键继续. . .
```

*通过指针引用多维数组

多维数组元素的地址

```
int a[3][4]={{1,3,5,7},{9,11,13,15},{17,19,21,23}};
```



表示形式	含义	地址
a	二维数组名，指向一维数组a[0]，即0行起始地址	2000
a[0], *(a+0), *a	0行0列元素地址	2000
a+1, &a[1]	指向第1行起始地址	2016
a[1], *(a+1)	1行0列元素a[1][0]的地址	2016
a[1]+2, *(a+1)+2, &a[1][2]	1行2列元素a[1][2]的地址	2024
*(a[1]+2), *(*(a+1)+2), a[1][2]	1行2列元素a[1][2]的值	是元素值，为13

多维数组元素的地址

C语言的地址信息中既包含位置信息(如内存编号2000)，还包含它所指向的数据的类型信息。

a[0]是一维数组名，它是一维数组中起始元素的地址，a是二维数组名，它是二维数组的首行起始地址，二者的纯地址是相同的，即2000，但它们的基类型不同，即它们指向的数据的类型不同，前者是整型数据，后者是一维数组。

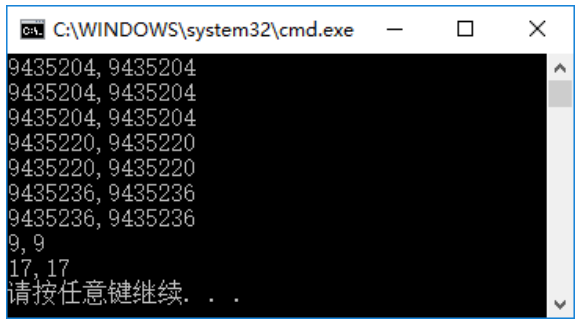
如果用一个指针变量pt来指向此一维数组，应当这样定义:

```
int (*pt)[4];

//表示pt指向由4个整型元素组成的一维数组，此时指针变量pt的基类型是由4个整型元素组成的一维数组
```

多维数组元素的地址

【例8.11】输出二维数组的有关数据(地址和元素的值)。

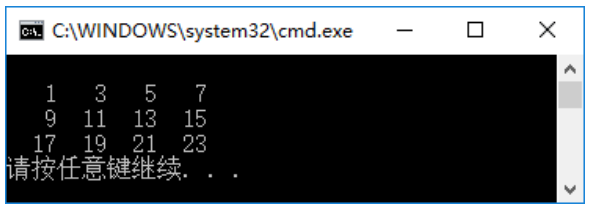
```
#include <stdio.h>
int main()
{    int a[3][4]={1,3,5,7,9,11,13,15,17,19,21,23};
     printf("%d,%d\n",a,*a);                 //0行起始地址和0行0列元素地址
     printf("%d,%d\n",a[0],*(a+0));          //0行0列元素地址
     printf("%d,%d\n",&a[0],&a[0][0]);       //0行起始地址和0行0列元素地址
     printf("%d,%d\n",a[1],a+1);             //1行0列元素地址和1行起始地址
     printf("%d,%d\n",&a[1][0],*(a+1)+0);    //1行0列元素地址
     printf("%d,%d\n",a[2],*(a+2));          //2行0列元素地址
     printf("%d,%d\n",&a[2],a+2);            //2行起始地址
     printf("%d,%d\n",a[1][0],*(*(a+1)+0));  //1行0列元素的值
     printf("%d,%d\n",*a[2],*(*(a+2)+0));    //2行0列元素的值
     return 0;
}
```



指向数组元素的指针变量

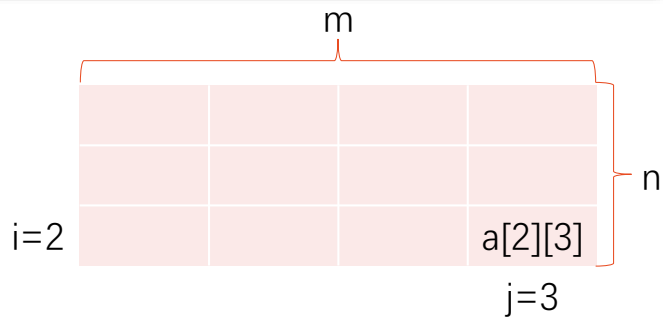
【例8.12】有一个3×4的二维数组，要求用指向元素的指针变量输出二维数组各元素的值。

```
#include <stdio.h>
int main()
{     int a[3][4]={1,3,5,7,9,11,13,15,17,19,21,23};
      int *p;                                  //p是int *型指针变量
      for(p=a[0];p<a[0]+12;p++)                //使p依次指向下一个元素
      {     if((p-a[0])%4==0) printf("\n");     //p移动4次后换行
            printf("%4d",*p);                  //输出p指向的元素的值
      }
      printf("\n");
      return 0;
}
```



p是一个int *型(指向整型数据)的指针变量，它可以指向一般的整型变量，也可以指向整型的数组元素。每次使p值加1，使p指向下一元素。

如果要输出某个指定的数值元素（例如a[2][3]），则应事先计算该元素在数组中的相对位置（即相对于数组起始位置的相对位移量）。计算a[i][j]在数组中的相对位置的计算公式为：**i*m + j**，其中，m为二维数组的列数（二维数组大小为n×m）。



指向由m个元素组成的一维数组的指针变量

【例8.13】输出二维数组任一行任一列元素的值。

```
#include <stdio.h>
int main()
{    int a[3][4]={1,3,5,7,9,11,13,15,17,19,21,23};      //定义二维数组a并初始化
     int (*p)[4],i,j;                //指针变量p指向包含4个整型元素的一维数组
     p=a;                            //p指向二维数组的0行
     printf("please enter row and colum:");
     scanf("%d,%d",&i,&j);           //输入要求输出的元素的行列号
     printf("a[%d,%d]=%d\n",i,j,*(*(p+i)+j));          //输出a[i][j]的值
     return 0;
}
```

```
C:\WINDOWS\system32\cmd.exe
please enter row and colum:1,2
a[1,2]=13
请按任意键继续. . .
```

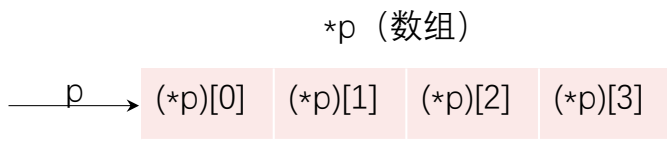
比较:

① int a[4];（a有4个元素，每个元素为整型）

② int (*p)[4];

第②种形式表示(*p)有4个元素，每个元素为整型。也就是p所指的对象是有4个整型元素的数组，即p是指向一维数组的指针，见图8.24。应该记住，此时p只能指向一个包含4个元素的一维数组，不能指向一维数组中的某一元素。p的值是该一维数组的起始地址。虽然这个地址（指纯地址）与该一维数组首元素的地址相同，但它们的基类型是不同的。

```
#include <stdio.h>
int main()
{    int a[4]={1,3,5,7};             //定义一维数组a，包含4个元素
     int (*p)[4];                    //定义指向包含4个元素的一维数组的指针变量中
     p=&a;                           //使p指向一维数组
     printf("%d\n",(*p)[3]);         //输出a[3]，输出整数7
     return 0;
}
```



指向由m个元素组成的一维数组的指针变量

要注意指针变量的类型，从“int (*p)[4];”可以看到，p的类型不是int *型，而是int (*)[4]型，p被定义为指向一维整型数组的指针变量，一维数组有4个元素，因此p的基类型是一维数组，其长度是16字节。“*(p+2)+3”括号中的2是以p的基类型(一维整型数组)的长度为单位的，即p每加1，地址就增加16个字节（4个元素，每个元素4个字节），而“*(p+2)+3”括号外的数字3，不是以p的基类型的长度为单位的。由于经过*(p+2)的运算，得到a[2]，即&a[2][0]，它已经转化为指向列元素的指针了，因此加3是以元素的长度为单位的，加3就是加(3×4)个字节。虽然p+2和*(p+2)具有相同的值，但由于它们所指向的对象的长度不同，因此(p+2)+3和*(p+2)+3的值就不相同了。

用指向数组的指针作函数参数

一维数组名可以作为函数参数，多维数组名也可作函数参数。

用**指针变量作形参**，以接受实参数组名传递来的地址。可以有两种方法:

① 用指向变量的指针变量；

② 用指向一维数组的指针变量。

用指向数组的指针作函数参数

【例8.14】有一个班，3个学生，各学4门课，计算总平均分数以及第n个学生的成绩。

```
#include <stdio.h>
int main()
{     void average(float *p,int n);
      void search(float (*p)[4],int n);
      float score[3][4]={{65,67,70,60},{80,87,90,81},{90,99,100,98}};
      average(*score,12);              //求12个分数的平均分
      search(score,2);                 //求序号为2的学生的成绩
      return 0;
}

void average(float *p,int n)           //定义求平均成绩的函数
{     float *p_end;
      float sum=0,aver;
      p_end=p+n-1;
      //n的值为12时，p_end的值是p+11，指向最后一个元素
      for(;p<=p_end;p++)
            sum=sum+(*p);
      aver=sum/n;
      printf("average=%5.2f\n",aver);
}

void search(float (*p)[4],int n)
//p是指向具有4个元素的一维数组的指针
{     int i;
      printf("The score of No.%d are:\n",n);
      for(i=0;i<4;i++)
            printf("%5.2f ",*(*(p+n)+i));
      printf("\n");
}
```

```
C:\WINDOWS\system32\cmd.exe
average=82.25
The score of No.2 are:
90.00 99.00 100.00 98.00
请按任意键继续. . .
```

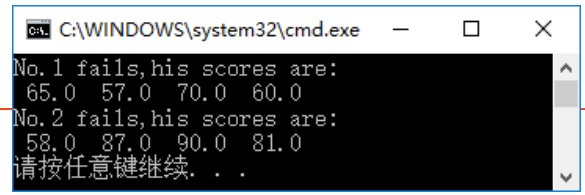
注意

- 实参与形参如果是指针类型，应当注意它们的**基类型必须一致**。不应把int *型的指针(即数组元素的地址)传给int (*)[4] 型(指向一维数组)的指针变量，反之亦然。

用指向数组的指针作函数参数

【例8.15】在例8.14的基础上，查找有一门以上课程不及格的学生，输出他们的全部课程的成绩。

```
#include <stdio.h>
int main()
{    void search(float (*p)[4],int n);      //函数声明
     float score[3][4]={{65,57,70,60},{58,87,90,81},{90,99,100,98}};
     //定义二维数组函数score
     search(score,3);                       //调用search函数
     return 0;
}

void search(float (*p)[4],int n)
//形参p是指向包含4个float型元素的一维数组的指针变量
{    int i,j,flag;
     for(j=0;j<n;j++)
     {    flag=0;
          for(i=0;i<4;i++)
               if(*(*(p+j)+i)<60) flag=1;
               //*(*(p+j)+i)就是score[j][i]
          if(flag==1)
          {    printf("No.%d fails,his scores are:\n",j+1);
               for(i=0;i<4;i++)
                    printf("%5.1f ",*(*(p+j)+i));
                    //输出*(*(p+j)+i)就是输出score[j][i]的值
               printf("\n");
          }
     }
}
```



通过指针引用字符串

字符串的引用方式

(1) 用字符数组存放一个字符串，可以通过数组名和下标引用字符串中一个字符，也可以通过数组名和格式声明“%s”输出该字符串。

(2) 用字符指针变量指向一个字符串常量，通过字符指针变量引用字符串常量。

字符串的引用方式

【例8.16】定义一个字符数组，在其中存放字符串"I love China!"，输出该字符串和第8个字符。

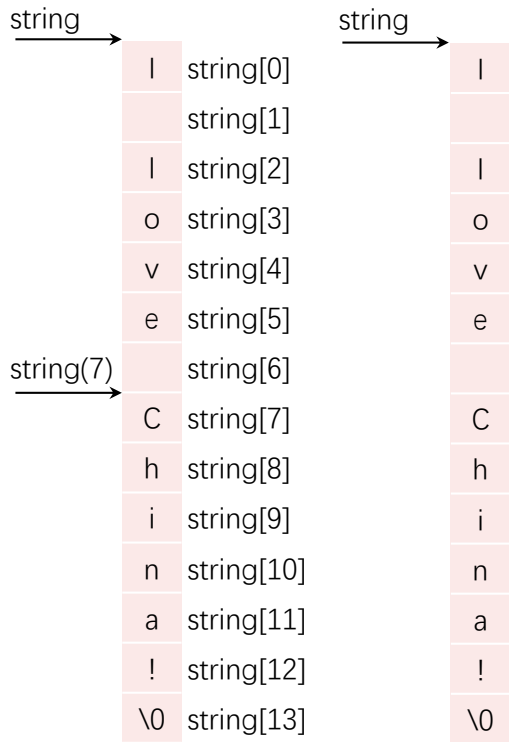
【例8.17】通过字符指针变量输出一个字符串。

```
#include <stdio.h>
int main()
{     char string[]="I love China!"; //定义字符数组sting
      printf("%s\n",string);          //用%s格式声明输出string，可以输出整个字符串
      printf("%c\n",string[7]);       //用%c格式输出一个字符数组元素
      return 0;
}
```

```
C:\WINDOWS\system32\cmd.exe
I love China!
C
请按任意键继续. . .
```

```
#include <stdio.h>
int main()
{     char *string="I love China!"; //定义字符指针变量string并初始化
      printf("%s\n",string);          //输出字符串
      return 0;
}
```

```
C:\WINDOWS\system32\cmd.exe
I love China!
请按任意键继续. . .
```



字符串的引用方式

在C语言中只有字符变量，没有字符串变量。

```
char *string="I love China!";
```

==

```
char *string;       //定义一个char *型变量
string="I love China!";
//把字符串第1个元素的地址赋给字符指针变量string
```

注意

- string被定义为一个指针变量，基类型为字符型。它只能指向一个字符类型数据，而不能同时指向多个字符数据，更不是把"I love China!"这些字符存放到string中（指针变量只能存放地址），也不是把字符串赋给*string。只是把"I love China!"的第1个字符的地址赋给指针变量string。

可以对指针变量进行再赋值，如：

```
string="I am a student.";       //对指针变量string重新赋值
```

可以通过字符指针变量输出它所指向的字符串，如：

```
printf("%s\n",string); //%s可对字符串进行整体的输入输出
```

%s是输出字符串时所用的格式符，在输出项中给出字符指针变量名string，则系统会输出string所指向的字符串第1个字符，然后自动使string加1，使之指向下一个字符，再输出该字符……如此直到遇到字符串结束标志'\0'为止。注意，在内存中，字符串的最后被自动加了一个'\0'。

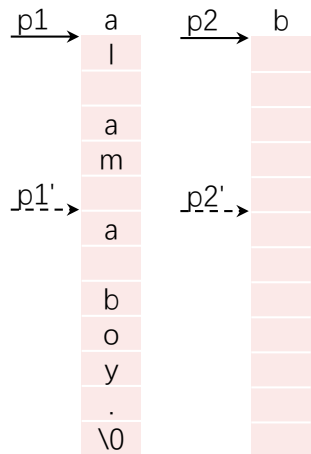
字符串的引用方式

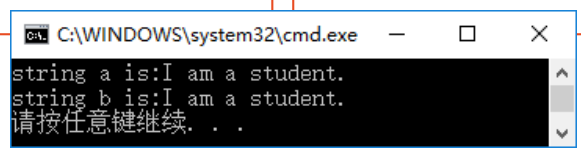
对字符串中字符的存取，可以用下标方法，也可以用指针方法。

【例8.18】将字符串a复制为字符串b，然后输出字符串b。【例8.19】用指针变量来处理例8.18问题。

```
#include <stdio.h>
int main()
{     char a[]="I am a student.",b[20];  //定义字符数组
      int i;
      for(i=0;*(a+i)!='\0';i++)
            *(b+i)=*(a+i);          //将a[i]的值赋给b[i]
      *(b+i)='\0';                  //在b数组的有效字符之后加'\0'
      printf("string a is:%s\n",a);//输出a数组中全部有效字符
      printf("string b is:");
      for(i=0;b[i]!='\0';i++)
            printf("%c",b[i]);      //逐个输出b数组中全部有效字符
      printf("\n");
      return 0;
}
```

```
#include <stdio.h>
int main()
{     char a[]="I am a boy.",b[20],*p1,*p2;
      p1=a;p2=b;
      //p1,p2分别指向a数组和b数组中的第一个元素
      for(;*p1!='\0';p1++,p2++)   //p1,p2每次自加1
            *p2=*p1;
            //将p1所指向的元素的值赋给p2所指向的元素
      *p2='\0';                   //在复制完全部有效字符后加'\0'
      printf("string a is:%s\n",a);     //输出a数组中的字符
      printf("string b is:%s\n",b);     //输出b数组中的字符
      return 0;
}
```



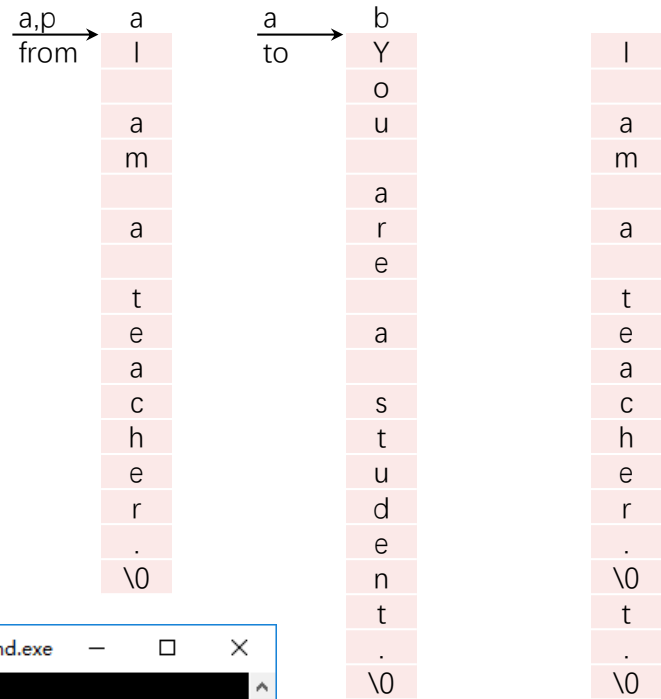


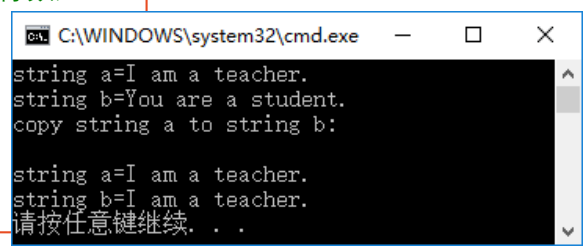
字符指针作函数参数

【例8.20】用函数调用实现字符串的复制。

(1) 用字符数组名作为函数参数

```
#include <stdio.h>
int main()
{     void copy_string(char from[], char to[]);
      char a[]="I am a teacher.";
      char b[]="You are a student.";
      printf("string a=%s\nstring b=%s\n",a,b);
      printf("copy string a to string b:\n");
      copy_string(a,b);          //用字符数组名作为函数实参
      printf("\nstring a=%s\nstring b=%s\n",a,b);
      return 0;
}

void copy_string(char from[], char to[])      //形参为字符数组
{     int i=0;
      while(from[i]!='\0')
      {     to[i]=from[i]; i++;}
      to[i]='\0';
}
```





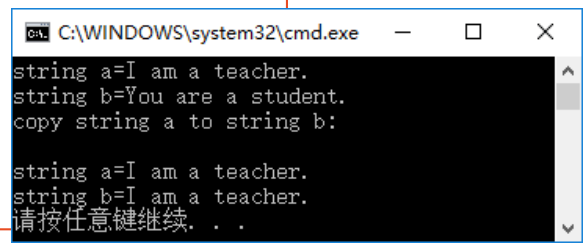
字符指针作函数参数

【例8.20】用函数调用实现字符串的复制。

(2) 用字符型指针变量作实参

```
#include <stdio.h>
int main()
{     void copy_string(char from[], char to[]);      //函数声明
      char a[]="I am a teacher.";          //定义字符数组a并初始化
      char b[]="You are a student.";      //定义字符数组b并初始化
      char *from=a,*to=b;     //from指向a数组首元素，to指向b数组首元素
      printf("string a=%s\nstring b=%s\n",a,b);
      printf("copy string a to string b:\n");
      copy_string(from,to);     //实参为字符指针变量
      printf("\nstring a=%s\nstring b=%s\n",a,b);
      return 0;
}
void copy_string(char from[], char to[])              //形参为字符数组
{     int i=0;
      while(from[i]!='\0')
      {     to[i]=from[i]; i++;}
      to[i]='\0';
}
```

指针变量from的值是a数组首元素的地址，指针变量to的值是b数组首元素的地址。它们作为实参，把a数组首元素的地址和b数组首元素的地址传递给形参数组名from和to(它们实质上也是指针变量)。其他与程序(1)相同。

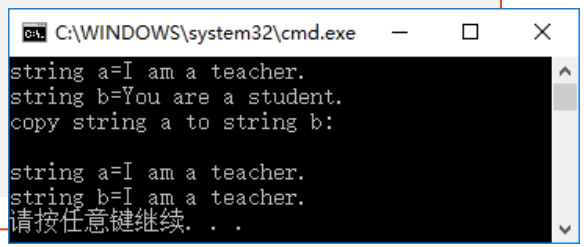


字符指针作函数参数

【例8.20】用函数调用实现字符串的复制。

(3) 用字符指针变量作形参和实参

```
#include <stdio.h>
int main()
{    void copy_string(char *from, char *to);
     char *a="I am a teacher.";          //a是char*型指针变量
     char b[]="You are a student.";      //b是字符数组
     char *p=b;           //使指针变量p指向b数组首元素
     printf("string a=%s\nstring b=%s\n",a,b);     //输出a串和b串
     printf("copy string a to string b:\n");
     copy_string(a,p);    //调用copy_string函数，实参为指针变量
     printf("\nstring a=%s\nstring b=%s\n",a,b);  //输出改变后的a串和b串
     return 0;
}
void copy_string(char *from, char *to) //定义函数，形参为字符指针变量
{    for(;*from!='\0';from++,to++)
     {     *to=*from;}
     *to='\0';
}
```



改进

```
void copy_string(char *from, char *to)
{    for(;(*to++=* from++)!='\0';);
     //或for(;*to++=* from++;);
}
```

```
void copy_string(char *from, char *to)
{    while((*to=*from)!='\0')
     //或while(*to=*from)
     {     to++; from++;}
}
```

```
void copy_string(char *from, char *to)
{    while(*from!='\0')
     //或while(*from)，因为'\0'的ASCII码为0
          *to++=*from++;
     *to='\0';
}
```

```
void copy_string(char *from, char *to)
{    while((*to++=*from++)!='\0');
     //或while(*to++=*from++)
}
```

```
void copy_string（char from[]，char to[]）
{    char *p1, *p2;
     p1=from;p2=to;
     while((*p2++=*p1++)!='\0');
}
```

字符指针作函数参数

字符指针作为函数参数时，实参与形参的类型有以下几种对应关系：

实参	形参
字符数组名	字符数组名
字符数组名	字符指针变量
字符指针变量	字符指针变量
字符指针变量	字符数组名

使用字符指针变量和字符数组的比较

(1) **字符数组由若干个元素组成，每个元素中放一个字符，而字符指针变量中存放的是地址**(字符串第1个字符的地址)，绝不是将字符串放到字符指针变量中。

(2) 赋值方式。**可以对字符指针变量赋值，但不能对数组名赋值。**(数组名是常量)

(3) 初始化的含义。

```
char *a="I love China!";
```
≡
```
char *a;
a="I love China!";
```

```
char str[14]="I love China!";
```
≠
```
char str[14];
str[]="I love China!";
```


(4) 存储单元的内容。**编译时为字符数组分配若干存储单元，以存放各元素的值，而对字符指针变量，只分配一个存储单元**(Visual C++为指针变量分配4个字节)。

```
char *a;
scanf("%s",a);
```


```
char *a,str[10];
a=str; scanf("%s",a);
```


(5) **指针变量的值是可以改变的，而字符数组名代表一个固定的值(数组首元素的地址)，不能改变。**

(6) 字符数组中各元素的值是可以改变的(可以对它们再赋值)，但字符指针变量指向的字符串常量中的内容是不可以被取代的(不能对它们再赋值)。

```
char a[]="House";
a[2]='r';
```


```
char *b="House";
b[2]='r';
```


(7) 引用数组元素。对字符数组可以用下标法(用数组名和下标)引用一个数组元素(如a[5])，也可以用地址法(如*(a+5))引用数组元素a[5]。如果定义了字符指针变量p，并使它指向数组a的首元素，则可以用指针变量带下标的形式引用数组元素(如p[5])，同样，可以用地址法(如*(p+5))引用数组元素a[5]。

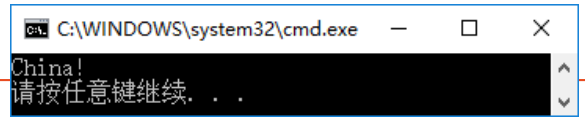
(8) 用指针变量指向一个格式字符串，可以用它代替printf函数中的格式字符串。

```
char *format="a=%d,b=%f\n";
printf(format,a,b);
```

使用字符指针变量和字符数组的比较

【例8.21】改变指针变量的值。

```
#include <stdio.h>
int main()
{    char *a="I love China!";
     a=a+7;              //改变指针变量的值，即改变指针变量的指向
     printf("%s\n",a);   //输出从a指向的字符开始的字符串
     return 0;
}
```




```
#include <stdio.h>
int main()
{    char str[]={"I love China!"};
     str=str+7;
     printf("%s\n",str);
     return 0;
}
```

指针变量a的值是可以变化的。printf函数输出字符串时，从指针变量a当时所指向的元素开始，逐个输出各个字符，直到遇'\0'为止。而数组名虽然代表地址，但它是常量，它的值是不能改变的。

指针变量的值是可以改变的，而字符数组名代表一个固定的值(数组首元素的地址)，不能改变。

*指向函数的指针

什么是函数的指针

如果在程序中定义了一个函数，在编译时会把函数的源代码转换为可执行代码并分配一段存储空间。这段内存空间有一个起始地址，也称为函数的入口地址。每次调用函数时都从该地址入口开始执行此段函数代码。

函数名就是函数的指针，它代表函数的起始地址。

可以定义一个指向函数的指针变量，用来存放某一函数的起始地址，这就意味着此指针变量指向该函数。例如：

```
int (*p)(int,int);
```

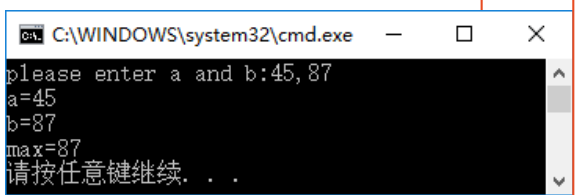
定义p是一个指向函数的指针变量，它可以指向函数类型为整型且有两个整型参数的函数。此时，指针变量p的类型用int (*)(int,int)表示。

用函数指针变量调用函数

【例8.22】用函数求整数a和b中的大者。

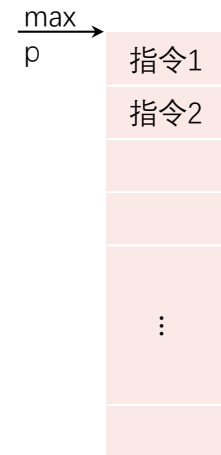
(1)通过函数名调用函数

```
#include <stdio.h>
int main()
{     int max(int,int);      //函数声明
      int a,b,c;
      printf("please enter a and b:");
      scanf("%d,%d",&a,&b);
      c=max(a,b);            //通过函数名调用max函数
      printf("a=%d\nb=%d\nmax=%d\n",a,b,c);
      return 0;
}

int max(int x,int y)         //定义max函数
{     int z;
      if(x>y) z=x;
      else z=y;
      return(z);
}
```



(2) 通过指针变量调用它所指向的函数

```
#include <stdio.h>
int main()
{     int max(int,int);      //函数声明
      int (*p)(int,int);     //定义指向函数的指针变量p
      int a,b,c;
      p=max;                 //使p指向max函数
      printf("please enter a and b:");
      scanf("%d,%d",&a,&b);
      c=(*p)(a,b);           //通过指针变量调用max函数
      printf("a=%d\nb=%d\nmax=%d\n",a,b,c);
      return 0;
}
int max(int x,int y)         //定义max函数
{     int z;
      if(x>y)z=x;
      else z=y;
      return(z);
}
```



怎样定义和使用指向函数的指针变量

类型名 (*指针变量名)(函数参数表列)

int (*p)(int,int);

(1) 定义指向函数的指针变量，并不意味着这个指针变量可以指向任何函数，它只能指向在定义时指定的类型的函数。

(2) 如果要用指针调用函数，必须先使指针变量指向该函数。

(3) 在给函数指针变量赋值时，只须给出函数名而不必给出参数。

(4) 用函数指针变量调用函数时，只须将(*p)代替函数名即可（p为指针变量名），在(*p)之后的括号中根据需要写上实参。

(5) 对指向函数的指针变量不能进行算术运算，如p+n,p++,p--等运算是无意义的。

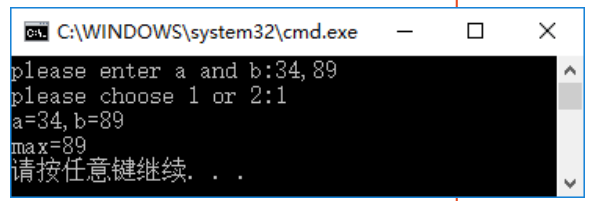
(6) 用函数名调用函数，只能调用所指定的一个函数，而通过指针变量调用函数比较灵活，可以根据不同情况先后调用不同的函数。

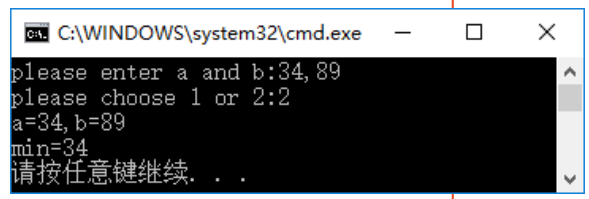
怎样定义和使用指向函数的指针变量

【例8.23】输入两个整数，然后让用户选择1或2，选1时调用max函数，输出二者中的大数，选2时调用min函数，输出二者中的小数。

```
#include <stdio.h>
int main()
{    int max(int,int);      //函数声明
     int min(int x,int y);  //函数声明
     int (*p)(int,int);     //定义指向函数的指针变量
     int a,b,c,n;
     printf("please enter a and b:");
     scanf("%d,%d",&a,&b);
     printf("please choose 1 or 2:");
     scanf("%d",&n);        //输入1或2
     if(n==1) p=max;    //如输入1，使p指向max函数
     else if (n==2) p=min;    //如输入2，使p指向min函数
     c=(*p)(a,b);           //调用p指向的函数
     printf("a=%d,b=%d\n",a,b);
     if(n==1) printf("max=%d\n",c);
     else printf("min=%d\n",c);
     return 0;
}
```

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

int min(int x,int y)
{    int z;
     if(x<y) z=x;
     else z=y;
     return(z);
}
```

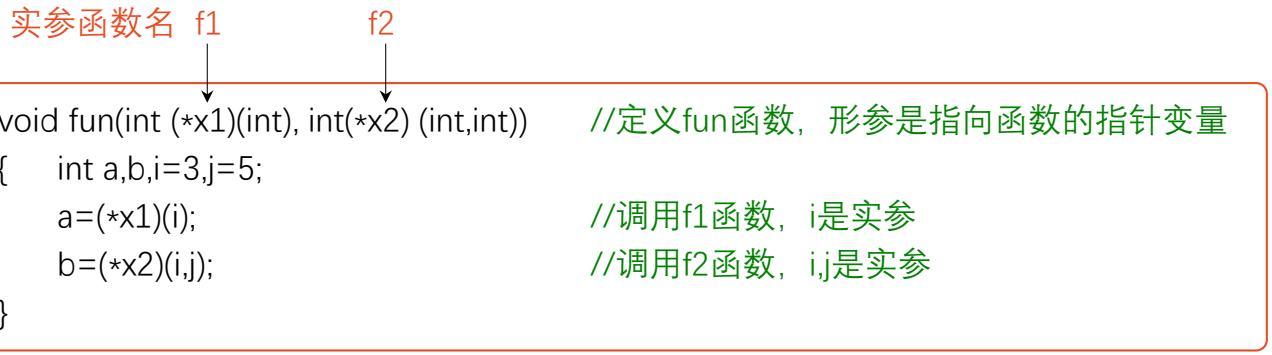




用指向函数的指针作函数参数

指向函数的指针变量的一个重要用途是把函数的入口地址作为参数传递到其他函数。

指向函数的指针可以作为函数参数，把函数的入口地址传递给形参，这样就能够在被调用的函数中使用实参函数。它的原理可以简述如下: 有一个函数（假设函数名为fun），它有两个形参（x1和x2），定义x1和x2为指向函数的指针变量。在调用函数fun时，实参为两个函数名f1和f2，给形参传递的是函数f1和f2的入口地址。这样在函数fun中就可以调用f1和f2函数了。

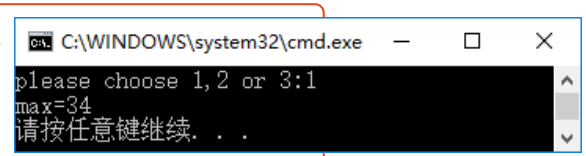


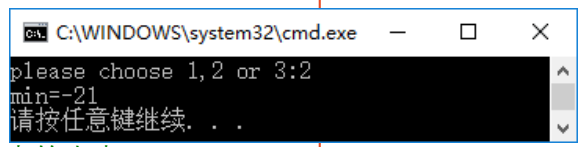
用指向函数的指针作函数参数

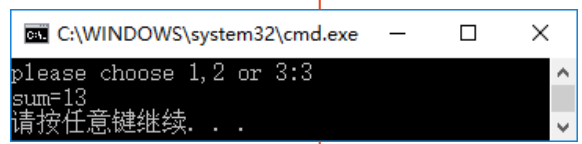
【例8.24】有两个整数a和b，由用户输入1,2或3。如输入1，程序就给出a和b中的大者，输入2，就给出a和b中的小者，输入3，则求a与b之和。

```
#include <stdio.h>
int main()
{    int fun(int x,int y, int (*p)(int,int)); //fun函数声明
     int max(int,int);                  //max函数声明
     int min(int,int);                  //min函数声明
     int add(int,int);                  //add函数声明
     int a=34,b=-21,n;
     printf("please choose 1,2 or 3:");
     scanf("%d",&n);                    //输入1,2或3之一
     if(n==1) fun(a,b,max);             //输入1时调用max函数
     else if(n==2) fun(a,b,min);        //输入2时调用min函数
     else if(n==3) fun(a,b,add);        //输入3时调用add函数
     return 0;
}

int fun(int x,int y,int (*p)(int,int))   //定义fun函数
{    int result;
     result=(*p)(x,y);
     printf("%d\n",result);             //输出结果
}
```

```
int max(int x,int y)  //定义max函数
{    int z;
     if(x>y) z=x;
     else z=y;
     printf("max=" );
     return(z);       //返回值是两数中的大者
}

int min(int x,int y)  //定义min函数
{    int z;
     if(x<y) z=x;
     else z=y;
     printf("min=");
     return(z);       //返回值是两数中的小者
}

int add(int x,int y)  //定义add函数
{    int z;
     z=x+y;
     printf("sum=");
     return(z);       //返回值是两数之和
}
```







*返回指针的函数

返回指针值的函数

类型名 *函数名(参数表列)

一个函数可以返回一个整型值、字符值、实型值等，也可以返回指针型的数据，即地址。其概念与以前类似，只是返回的值的类型是指针类型而已。

```
int *a(int x,int y);
```

a是函数名，调用它以后能得到一个int*型(指向整型数据)的指针，即整型数据的地址。x和y是函数a的形参，为整型。

注意

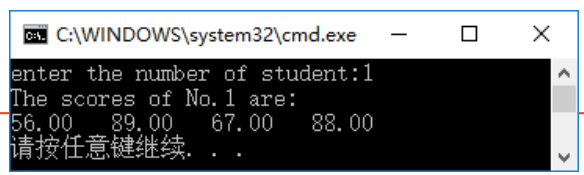
在“*a”两侧没有括号，在a的两侧分别为*运算符和()运算符。而()优先级高于*，因此a先与()结合，显然这是函数形式。这个函数前面有一个*，表示此函数是指针型函数（函数值是指针）。最前面的int表示返回的指针指向整型变量。

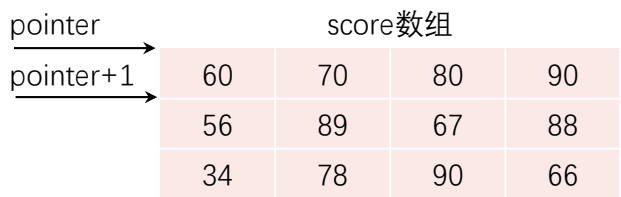
返回指针值的函数

【例8.25】有a个学生，每个学生有b门课程的成绩。要求在用户输入学生序号以后，能输出该学生的全部成绩。用指针函数来实现。

```
#include <stdio.h>
int main()
{    float score[][4]={{60,70,80,90},{56,89,67,88},{34,78,90,66}};
     //定义数组，存放成绩
     float *search(float (*pointer)[4],int n);  //函数声明
     float *p;
     int i,k;
     printf("enter the number of student:");
     scanf("%d",&k);      //输入要找的学生的序号
     printf("The scores of No.%d are:\n",k);
     p=search(score,k); //调用search函数，返回score[k][0]的地址
     for(i=0;i<4;i++)
          printf("%5.2f\t",*(p+i));   //输出score[k][0]~score[k][3]的值
     printf("\n");
     return 0;
}
```

```
float *search(float (*pointer)[4],int n)
//形参pointer是指向一维数组的指针变量
{    float *pt;
     pt=*(pointer+n);    //pt的值是&score[k][0]
     return(pt);
}
```



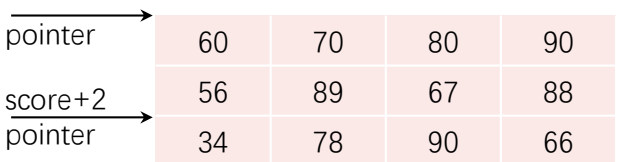


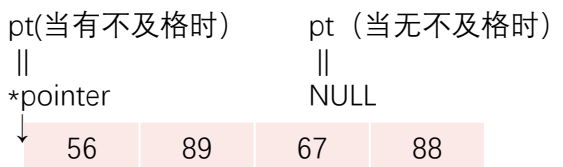
返回指针值的函数

【例8.26】对例8.25中的学生，找出其中有不及格的课程的学生及其学生号。

```
#include <stdio.h>
int main()
{    float score[][4]={{60,70,80,90},{56,89,67,88},{34,78,90,66}};
     //定义数组，存放成绩
     float *search(float (*pointer)[4]);  //函数声明
     float *p;
     int i,j;
     for(i=0;i<3;i++)                    //循环3次
     {    p=search(score+i);
//调用search函数,如有不及格返回score[i][0]的地址,否则返回NULL
          if(p==*(score+i))
          //如果返回的是score[i][0]的地址，表示p的值不是NULL
          {    printf("No.%d score:",i);
               for(j=0;j<4;j++)
                    printf("%5.2f  ",*(p+j));
                    //输出score[i][0]~score[i][3]的值
               printf("\n");
          }
     }
     return 0;
}

float *search(float (*pointer)[4])
//定义函数，形参pointer是指向一维数组的指针变量
{    int i=0;
     float *pt;
     pt=NULL;      //先使pt的值为NULL
     for(;i<4;i++)
          if(*(*pointer+i)<60) pt=*pointer;
          //如果有不及格课程，使pt指向score[i][0]
     return(pt);
}
```

```
C:\WINDOWS\system32\cmd.exe
No.1 score:56.00  89.00  67.00  88.00
No.2 score:34.00  78.00  90.00  66.00
请按任意键继续. . .
```





*指针数组和多重指针

什么是指针数组

定义一维指针数组的一般形式为 **类型名 *数组名[数组长度];**

```
int *p[4];
```

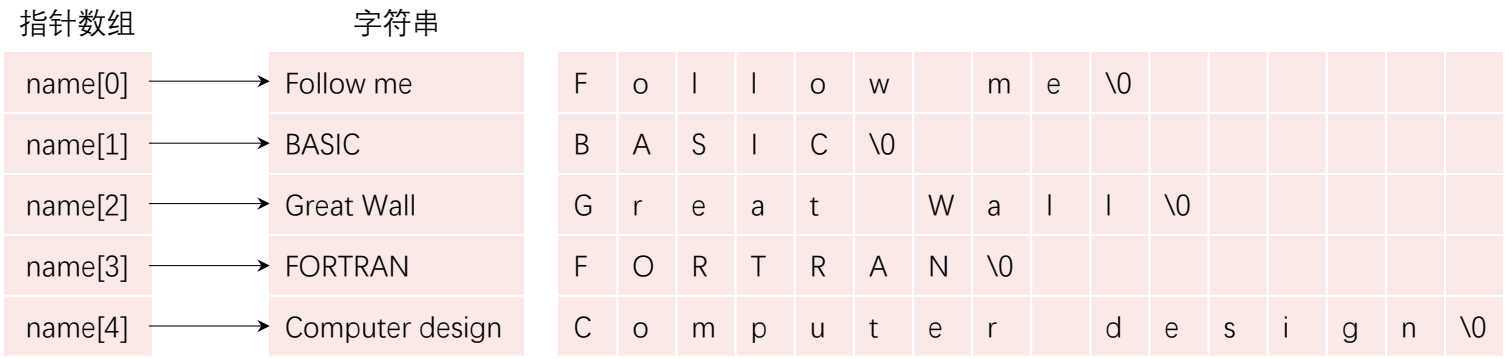
一个数组，若其元素均为指针类型数据，称为**指针数组**，也就是说，指针数组中的每一个元素都存放一个地址，相当于一个指针变量。

指针数组比较适合用来指向若干个字符串，使字符串处理更加方便灵活。

什么是指针数组

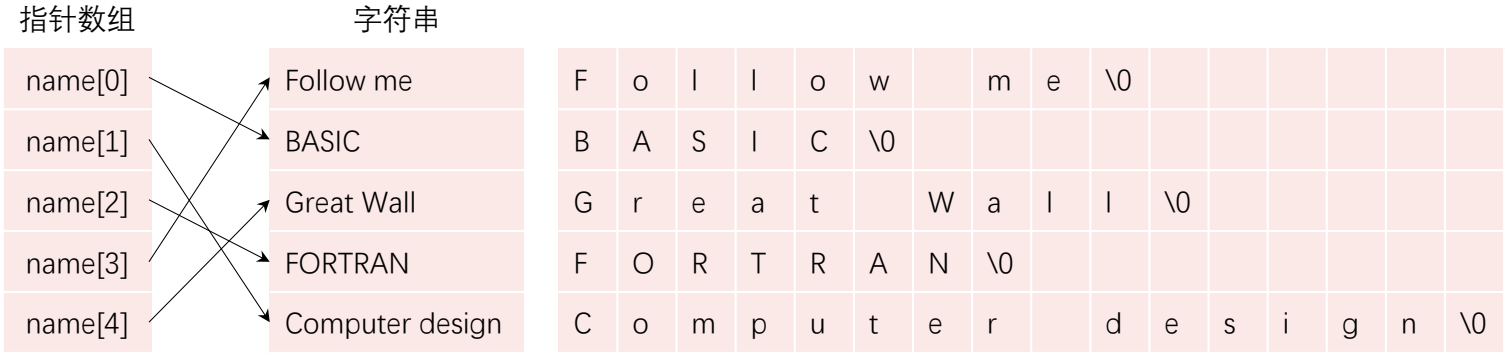
【例8.27】将若干字符串按字母顺序（由小到大）输出。

排序前



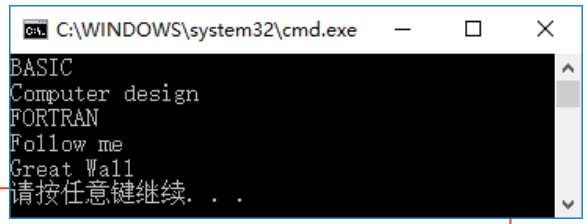
指向互换

排序后



什么是指针数组

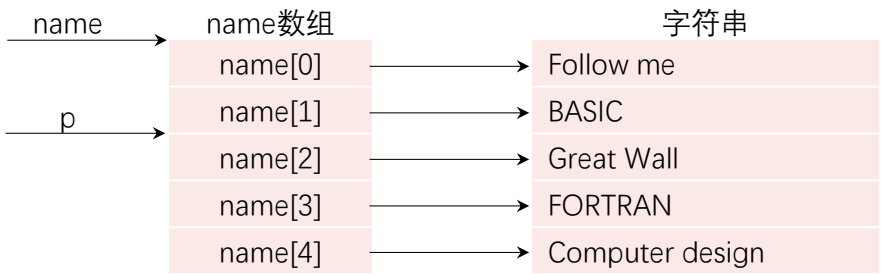
【例8.27】将若干字符串按字母顺序（由小到大）输出。




```
#include <stdio.h>
#include <string.h>
int main()
{     void sort(char *name[],int n);      //函数声明
      void print(char *name[],int n);     //函数声明
      char *name[]={"Follow me","BASIC",
      "Great Wall","FORTRAN","Computer design"};
      //定义指针数组，它的元素分别指向5个字符串
      int n=5;
      sort(name,n); //调用sort函数，对字符串排序
      print(name,n);//调用print函数，输出字符串
      return 0;
}

void sort(char *name[],int n)            //定义sort函数
{     char *temp;
      int i,j,k;
      for(i=0;i<n-1;i++)                 //用选择法排序
      {     k=i;
            for(j=i+1;j<n;j++)
                  if(strcmp(name[k],name[j])>0) k=j;
            if(k!=i)
            {     temp=name[i]; name[i]=name[k]; name[k]=temp;}
      }
}

void print(char *name[],int n)      //定义print函数
{     int i;
      for(i=0;i<n;i++)
            printf("%s\n",name[i]);
            //按指针数组元素的顺序输出它们所指向的字符串
}
```

指向指针数据的指针变量

在了解了指针数组的基础上，需要了解**指向指针数据的指针变量**，简称为**指向指针的指针**。



name是一个指针数组，它的每一个元素是一个指针型的变量，其值为地址。name既然是一个数组，它的每一元素都应有相应的地址。数组名name代表该指针数组首元素的地址。name+i是name[i]的地址。name+i就是指向指针型数据的指针。还可以设置一个指针变量p，它指向指针数组的元素。p就是指向指针型数据的指针变量。

定义一个指向指针数据的指针变量:

```
char **p;
```

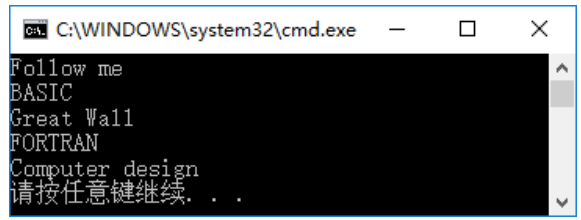
p的前面有两个*号。p指向一个字符指针变量（这个字符指针变量指向一个字符型数据）。如果引用*p，就得到p所指向的字符指针变量的值。

```
p=name+2;
printf("%d\n",*p);      //name[2]的值（它是一个地址）
printf("%s\n",*p);      //以字符串形式(%s)输出字符串"Great Wall"
```

指向指针数据的指针变量

【例8.28】使用指向指针数据的指针变量。

```
#include <stdio.h>
int main()
{    char *name[]={"Follow me","BASIC","Great Wall","FORTRAN","Computer design"};
     char **p;
     int i;
     for(i=0;i<5;i++)
     {    p=name+i;
          printf("%s\n",*p);
     }
     return 0;
}
```



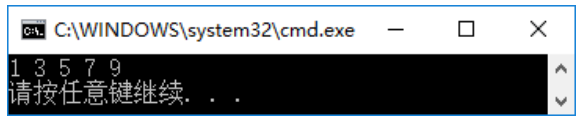
p是指向char*型数据的指针变量，即指向指针的指针。在第1次执行for循环体时，赋值语句“p=name+i;”使p指向name数组的0号元素name［0］，*p是name［0］的值，即第1个字符串首字符的地址，用printf函数输出第1个字符串（格式符为%s）。执行5次循环体，依次输出5个字符串。

指针数组的元素也可以不指向字符串，而指向整型数据或实型数据等。

指向指针数据的指针变量

【例8.29】有一个指针数组，其元素分别指向一个整型数组的元素，用指向指针数据的指针变量，输出整型数组各元素的值。

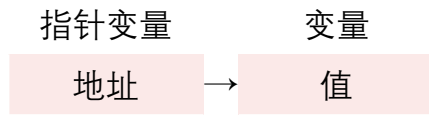
```
#include <stdio.h>
int main()
{    int a[5]={1,3,5,7,9};
     int *num[5]={&a[0],&a[1],&a[2],&a[3],&a[4]};
     int **p,i;                    //p是指向指针型数据的指针变量
     p=num;                        //使p指向num[0]
     for(i=0;i<5;i++)
     {    printf("%d ",**p);
          p++;
     }
     printf("\n");
     return 0;
}
```



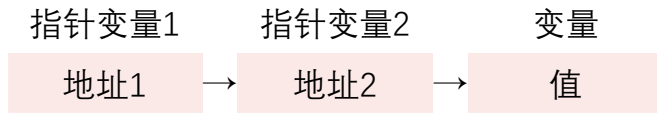
指向指针数据的指针变量

利用指针变量访问另一个变量就是“间接访问”。

如果在一个指针变量中存放一个目标变量的地址，这就是“单级间址”；



指向指针数据的指针用的是“二级间址”方法；



从理论上说，间址方法可以延伸到更多的级，即多重指针。

指针数组作main函数的形参

指针数组的一个重要应用是作为main函数的形参。在以往的程序中，main函数的第1行一般写成以下形式：`int main()` 或 `int main(void)`

括号中是空的或有“void”，表示main函数**没有参数，调用main函数时不必给出实参**。

在某些情况下，main函数可以有参数，即：`int main(int argc, char *argv[])`

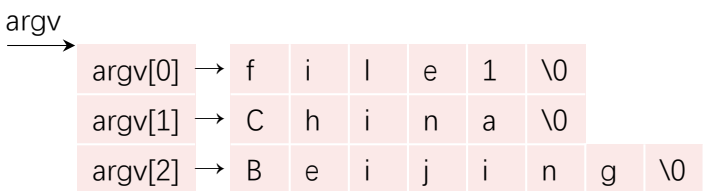
其中，argc和argv就是main函数的形参，它们是程序的“命令行参数”。argc(argument count的缩写，意思是参数个数)，argv(argument vector缩写，意思是参数向量)，它是一个*char指针数组，数组中每一个元素(其值为指针)指向命令行中的一个字符串的首字符。

注意 如果用带参数的main函数，其第一个形参必须是int型，用来接收形参个数（文件名也算一个参数），第二个形参必须是字符指针数组，用来接收从操作系统命令行传来的字符串中首字符的地址。

main函数是操作系统调用的，实参只能由操作系统给出。在操作命令状态下，实参是和执行文件的命令一起给出的。

命令名 参数1 参数2…参数n

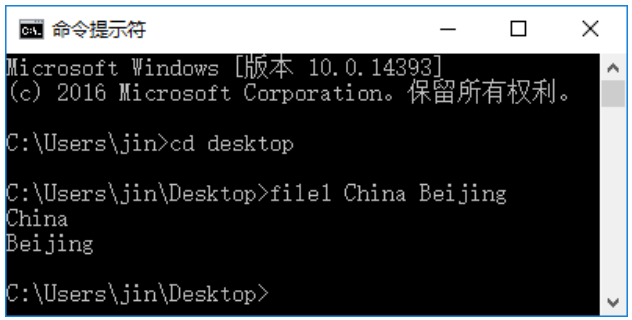
file1 China Beijing



指针数组作main函数的形参

```
int main(int argc,char *argv[])
{       while(argc>1)
        {       ++argv;
                printf("%s\n", *argv);
                --argc;
        }
        return 0;
}
```

```
int main(int argc,char *argv[])
{       while(argc-->1)
                printf("%s\n", *++argv);
        return 0;
}.
```



*动态内存分配与指向它的指针变量

什么是内存的动态分配

全局变量是分配在内存中的静态存储区的，非静态的局部变量(包括形参)是分配在内存中的动态存储区的，这个存储区是一个称为**栈**(stack)的区域。除此以外，C语言还允许建立内存动态分配区域，以存放一些临时用的数据，这些数据不必在程序的声明部分定义，也不必等到函数结束时才释放，而是需要时随时开辟，不需要时随时释放。这些数据是临时存放在一个特别的自由存储区，称为**堆**(heap)区。可以根据需要，向系统申请所需大小的空间。由于未在声明部分定义它们为变量或数组，因此不能通过变量名或数组名去引用这些数据，只能通过指针来引用。

怎样建立内存的动态分配

用malloc函数开辟动态存储区

函数原型为 `void *malloc(unsigned int size);`

作用是在内存的动态存储区中分配一个长度为size的连续空间。形参size的类型定为无符号整型(不允许为负数)。此函数的值（即“返回值”）是所分配区域的第一个字节的地址，或者说，此函数是一个指针型函数，返回的指针指向该分配域的第一个字节。

```
malloc(100);      //开辟100字节的临时分配域，函数值为其第1个字节的地址
```

指针的基类型为void，即不指向任何类型的数据，只提供一个纯地址。如果此函数未能成功地执行(例如内存空间不足)，则返回空指针(NULL)。

怎样建立内存的动态分配

用calloc函数开辟动态存储区

函数原型为 `void *calloc(unsigned n, unsigned size);`

作用是在内存的动态存储区中分配n个长度为size的连续空间，这个空间一般比较大，足以**保存一个数组**。

用calloc函数可以为一维数组开辟动态存储空间，n为数组元素个数，每个元素长度为size。这就是动态数组。函数返回指向所分配域的第一个字节的指针；如果分配不成功，返回NULL。

```
p=calloc(50,4);        //开辟50×4个字节的临时分配域，把首地址赋给指针变量p
```

怎样建立内存的动态分配

用realloc函数重新分配动态存储区

函数原型为 **void *realloc(void *p,unsigned int size);**

如果已经通过malloc函数或calloc函数获得了动态空间，想改变其大小，可以用realloc函数重新分配。

用realloc函数将p所指向的动态空间的大小改变为size。p的值不变。如果重分配不成功，返回NULL。

```
realloc(p,50);      //将p所指向的已分配的动态空间改为50字节
```

怎样建立内存的动态分配

用free函数释放动态存储区

函数原型为 **void free(void *p);**

作用是释放指针变量p所指向的动态空间，使这部分空间能重新被其他变量使用。p应是最近一次调用calloc或malloc函数时得到的函数返回值。

```
free(p);    //释放指针变量p所指向的已分配的动态空间
```

free函数无返回值。

以上4个函数的声明在stdlib.h头文件中，在用到这些函数时应当用“#include <stdlib.h>”指令把stdlib.h头文件包含到程序文件中。

void指针类型

C 99允许使用基类型为void的指针类型。可以定义一个基类型为void的指针变量(即void*型变量)，它不指向任何类型的数据。在将它的值赋给另一指针变量时由系统对它进行类型转换，使之适合于被赋值的变量的类型。

注意

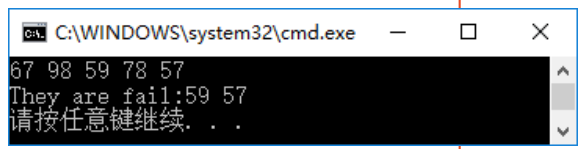
不要把“指向void类型”理解为能指向“任何的类型”的数据，而应理解为“指向空类型”或“不指向确定的类型”的数据。

```
int *pt;
pt=(int *)mcaloc(100);          //mcaloc(100)是void *型，把它转换为int *型
```

void指针类型

【例8.30】建立动态数组，输入5个学生的成绩，另外用一个函放数检查其中有无低于60分的，输出不合格的成绩。

```
#include <stdio.h>
#include <stdlib.h>                    //程序中用了malloc函数，应包含stdlib.h
int main()
{     void check(int *);               //函数声明
      int *p1,i;                       //p1是int型指针
      p1=(int *)malloc(5*sizeof(int)); //开辟动态内存区，将地址转换成int *型，然后放在p1中
      for(i=0;i<5;i++)
            scanf("%d",p1+i);          //输入5个学生的成绩
      check(p1);                       //调用check函数
      return 0;
}

void check(int *p)                     //定义check函数，形参是int*指针
{     int i;
      printf("They are fail:");
      for(i=0;i<5;i++)
            if(p[i]<60) printf("%d ",p[i]); //输出不合格的成绩
      printf("\n");
}
```



有关指针的小结

(1) 首先要准确理解指针的含义。“指针”是C语言中一个形象化的名词，形象地表示“指向”的关系，其在物理上的实现是通过地址来完成的。

- &a是变量a的地址，也可称为变量a的指针。
- 指针变量是存放地址的变量，也可以说，指针变量是存放指针的变量。
- 指针变量的值是一个地址，也可以说，指针变量的值是一个指针。
- 指针变量也可称为地址变量，它的值是地址。
- &是取地址运算符，&a是a的地址，也可以说，&是取指针运算符。&a是变量a的指针（即指向变量a的指针）。
- 数组名是一个地址，是数组首元素的地址，也可以说，数组名是一个指针，是数组首元素的指针。
- 函数名是一个指针(指向函数代码区的首字节)，也可以说函数名是一个地址(函数代码区首字节的地址)。
- 函数的实参如果是数组名，传递给形参的是一个地址，也可以说，传递给形参的是一个指针。

有关指针的小结

(2) 一个地址型的数据实际上包含3个信息：

① 表示内存编号的纯地址。

② 它本身的类型，即指针类型。

③ 以它为标识的存储单元中存放的是什么类型的数据，即基类型。

```
int a;
/* &a为a的地址，它就包括以上3个信息，它代表的是一个整型数据的地址，int是&a的基类型(即它指向的是int型的存储单元)。&a就是“指向整型数据的指针类型”或“基类型为整型的指针类型”，其类型可以表示为“int *”型。*/
```

有关指针的小结

(3) 要区别指针和指针变量。指针就是地址，而指针变量是用来存放地址的变量。

(4) 什么叫“指向”？地址就意味着指向，因为通过地址能找到具有该地址的对象。对于指针变量来说，把谁的地址存放在指针变量中，就说此指针变量指向谁。

注意

并不是任何类型数据的地址都可以存放在同一个指针变量中的，只有与指针变量的基类型相同的数据的地址才能存放在相应的指针变量中。

```
int a,*p;       //p是int*型的指针变量，基类型是int型
float b;
p=&a;           //a是int型，合法
p=&b;           //b是float型，类型不匹配
```

void *指针是一种特殊的指针，不指向任何类型的数据。如果需要用此地址指向某类型的数据，应先对地址进行类型转换。

有关指针的小结

(5) 要深入掌握在对数组的操作中正确地使用指针，搞清楚指针的指向。

```
int *p, a[10];      //p是指向int型类型的指针变量
p=a;                //p指向a数组的首元素
```

有关指针的小结

(6) 有关指针变量的归纳比较

变量定义	类型表示	含义
int i;	int	定义整型变量i
int *p;	int *	定义p为指向整型数据的指针变量
int a[5];	int [5]	定义整型数组a，它有5个元素
int *p[4];	int *[4]	定义指针数组p，它由4个指向整型数据的指针元素组成
int (*p)[4];	int (*)[4]	p为指向包含4个元素的一维数组的指针变量
int f();	int ()	f为返回整型函数值的函数
int *p();	int *()	p为返回一个指针的函数，该指针指向整型数据
int (*p)();	int (*)()	p为指向函数的指针，该函数返回一个整型值
int **p;	int **	p是一个指针变量，它指向一个指向整型数据的指针变量
void *p;	void *	p是一个指针变量，基类型为void(空类型)，不指向具体的对象

有关指针的小结

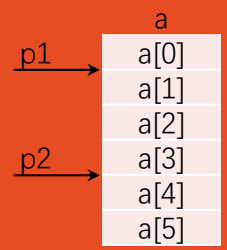
(7) 指针运算

① 指针变量加（减）一个整数。

```
p++;          //将该指针变量的原值(是一个地址)和它指向的变量所占用的存储单元的字节数相加
```

② 指针变量赋值。将一个变量地址赋给一个指针变量。 不应把一个整数赋给指针变量。

```
p=&a;         //将变量a的地址赋给p
p=array;      //将数组array首元素地址赋给p
p=&array[i];  //将数组array第i个元素的地址赋给p
p=max;        //max为已定义的函数，将max的入口地址赋给p
p1=p2;        //p1和p2是基类型相同指针变量，将p2的值赋给p1
```



③ 两个指针变量可以相减。如果两个指针变量都指向同一个数组中的元素，则两个指针变量值之差是两个指针之间的元素个数。

④ 两个指针变量比较。若两个指针指向同一个数组的元素，则可以进行比较。指向前面的元素的指针变量“小于”指向后面元素的指针变量。如果p1和p2不指向同一数组则比较无意义。

有关指针的小结

(8) 指针变量可以有空值，即该指针变量不指向任何变量。 p=NULL;

NULL是一个符号常量，代表整数0。在stdio.h头文件中对NULL进行了定义：#define NULL 0

它使p指向地址为0的单元。系统保证使该单元不作它用（不存放有效数据）。

注意 p的值为NULL与未对p赋值是两个不同的概念。前者是有值的（值为0），不指向任何变量，后者虽未对p赋值但并不等于p无值，只是它的值是一个无法预料的值，也就是p可能指向一个事先未指定的单元。

任何指针变量或地址都可以与NULL作相等或不相等的比较。 if(p==NULL)

有关指针的小结

指针的优点：

① 提高程序效率；

② 在调用函数时当指针指向的变量的值改变时，这些值能够为主调函数使用，即可以从函数调用得到多个可改变的值；

③ 可以实现动态存储分配。

如果使用指针不当，会出现隐蔽的、难以发现和排除的故障。因此，使用指针要十分小心谨慎。

第 9 章

用户自己建立数据类型

定义和使用结构体变量

自己建立结构体类型

struct 结构体名
{成员表列};

C语言允许用户自己建立由不同类型数据组成的组合型的数据结构，它称为**结构体**（structre）。

在程序中建立一个结构体类型：

num	name	sex	age	score	addr
10010	Li Fang	M	18	87.5	Beijing

```
struct Student
{    int num;            //学号为整型
     char name[20];      //姓名为字符串
     char sex;           //性别为字符型
     int age;            //年龄为整型
     float score;        //成绩为实型
     char addr[30];      //地址为字符串
};                       //注意最后有一个分号
```

结构体类型的名字是由一个关键字**struct**和结构体名组合而成的。结构体名由用户指定，又称“结构体标记”(structure tag)。

花括号内是该结构体所包括的子项，称为结构体的成员(member)。对各成员都应进行类型声明，即 **类型名 成员名;**

“成员表列”(member list)也称为“域表”(field list)，每一个成员是结构体中的一个域。成员名命名规则与变量名相同。

自己建立结构体类型

(1) 结构体类型并非只有一种，而是可以设计出许多种结构体类型，各自包含不同的成员。

(2) 成员可以属于另一个结构体类型。

num	name	sex	age	birthday			addr
				month	day	year	

```
struct Date                 //声明一个结构体类型 struct Date
{    int month;             //月
     int day;               //日
     int year;              //年
};
```

```
struct Student              //声明一个结构体类型 struct Student
{ int num;
char name[20];
char sex;
int age;
struct Date birthday;       //成员birthday属于struct Date类型
char addr[30];
};
```

定义结构体类型变量

1. 先声明结构体类型，再定义该类型的变量

```
struct Student
{    int num;              //学号为整型
     char name[20];        //姓名为字符串
     char sex;             //性别为字符型
     int age;              //年龄为整型
     float score;          //成绩为实型
     char addr[30];        //地址为字符串
};                         //注意最后有一个分号
```

```
struct Student  student1, student2;
       |            |          |
结构体类型名    结构体变量名
```

sutdent1:	10001	Zhang Xin	M	19	90.5	Shanghai
student2:	10002	Wang Li	F	20	98	Beijing

2. 在声明类型的同时定义变量

```
struct Student
{    int num;
     char name[20];
     char sex;
     int age;
     float score;
     char addr[30];
}student1, student2;
```

struct 结构体名
{ 成员表列
}变量名表列;

3. 不指定类型名而直接定义结构体类型变量

struct
{ 成员表列
}变量名表列;

定义结构体类型变量

(1) 结构体类型与结构体变量是不同的概念，不要混淆。只能对变量赋值、存取或运算，而不能对一个类型赋值、存取或运算。在编译时，对类型是不分配空间的，只对变量分配空间。

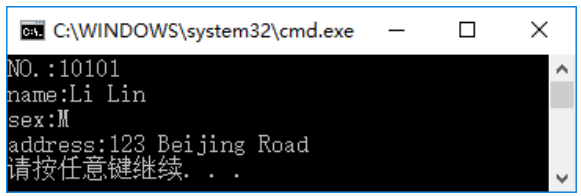
(2) 结构体类型中的成员名可以与程序中的变量名相同，但二者不代表同一对象。

(3) 对结构体变量中的成员（即“域”），可以单独使用，它的作用与地位相当于普通变量。

结构体变量的初始化和引用

【例9.1】把一个学生的信息(包括学号、姓名、性别、住址)放在一个结构体变量中，然后输出这个学生的信息。

```
#include <stdio.h>
int main()
{     struct Student                          //声明结构体类型struct Student
      {     long int num;                     //以下4行为结构体的成员
            char name[20];
            char sex;
            char addr[20];
      }a={10101,"Li Lin",'M',"123 Beijing Road"};  //定义结构体变量a并初始化
      printf("NO.:%ld\nname:%s\nsex:%c\naddress:%s\n",a.num,a.name,a.sex,a.addr);
      return 0;
}
```



结构体变量的初始化和引用

(1) 在定义结构体变量时可以对它的成员初始化。初始化列表是用花括号括起来的一些常量，这些常量依次赋给结构体变量中的各成员。

注意 对结构体变量初始化，不是对结构体类型初始化

(2) 可以引用结构体变量中成员的值，引用方式为 **结构体变量名.成员名**

```
student1.num=10010;
/*已定义了student1为student类型的结构体变量，则student1.num表示student1
变量中的num成员，即student1的num(学号)成员*/
```

“.”是成员运算符，它在所有的运算符中优先级最高，因此可以把student1.num作为一个整体来看待，相当于一个变量。

```
printf("%s\n",student1);          //企图用结构体变量名输出所有成员的值
```

不能企图通过输出结构体变量名来达到输出结构体变量所有成员的值。只能对结构体变量中的各个成员分别进行输入和输出。

结构体变量的初始化和引用

(3) 如果成员本身又属一个结构体类型，则要用若干个成员运算符，一级一级地找到最低的一级的成员。只能对最低级的成员进行赋值或存取以及运算。

```
student1.num=10010;              //结构体变量student1中的成员num
student1.birthday.month=6;       //结构体变量student1中的成员birthday中的成员month
```

(4) 对结构体变量的成员可以像普通变量一样进行各种运算（根据其类型决定可以进行的运算）。

```
student2.score = student1.score;          //赋值运算
sum=student1.score+student2.score;        //加法运算
student1.age++;                           //自加运算
```

(5) 同类的结构体变量可以互相赋值。

```
student1=student2;              //假设student1和student2已定义为同类型的结构体变量
```

(6) 可以引用结构体变量成员的地址，也可以引用结构体变量的地址(结构体变量的地址主要用作函数参数，传递结构体变量的地址)。但不能用以下语句整体读入结构体变量。

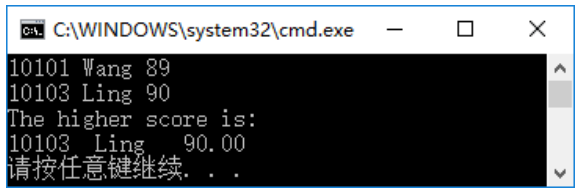
```
scanf("%d",&student1.num);      //输入student1.num的值
printf("%o",&student1);         //输出结构体变量student1的起始地址
```

```
scanf("%d,%s,%c,%d,%f,%s\n",&student1);
```

结构体变量的初始化和引用

【例9.2】输入两个学生的学号、姓名和成绩，输出成绩较高的学生的学号、姓名和成绩。

```
#include <stdio.h>
int main()
{    struct Student              //声明结构体类型struct Student
     {    int num;
          char name[20];
          float score;
     }student1,student2;      //定义两个结构体变量student1,student2
     scanf("%d%s%f",&student1.num,student1.name,&student1.score);    //输入学生1的数据
     scanf("%d%s%f",&student2.num,student2.name,&student2.score);    //输入学生1的数据
     printf("The higher score is:\n");
     if(student1.score>student2.score)
          printf("%d  %s  %6.2f\n",student1.num,student1.name,student1.score);
     else if(student1.score<student2.score)
          printf("%d  %s  %6.2f\n",student2.num,student2.name,student2.score);
     else
     {    printf("%d  %s  %6.2f\n",student1.num,student1.name,student1.score);
          printf("%d  %s  %6.2f\n",student2.num,student2.name,student2.score);
     }
     return 0;
}
```

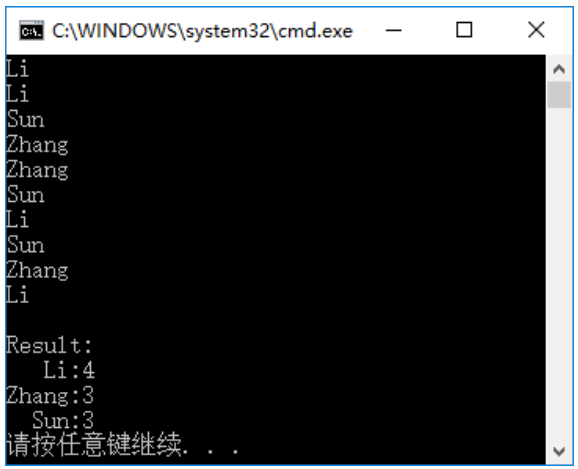


使用结构体数组

定义结构体数组

【例9.3】有3个候选人，每个选民只能投票选一人，要求编一个统计选票的程序，先后输入被选人的名字，最后输出各人得票结果。

name	count
Li	0
Zhang	0
Sun	0




```
#include <string.h>
#include <stdio.h>
struct Person                          //声明结构体类型struct Person
{     char name[20];                   //候选人姓名
      int count;                       //候选人得票数
}leader[3]={"Li",0,"Zhang",0,"Sun",0}; //定义结构体数组并初始化

int main()
{     int i,j;
      char leader_name[20];            //定义字符数组
      for(i=1;i<=10;i++)
      {     scanf("%s",leader_name);   //输入所选的候选人姓名
            for(j=0;j<3;j++)
            if(strcmp(leader_name,leader[j].name)==0) leader[j].count++;
      }
      printf("\nResult:\n");
      for(i=0;i<3;i++)
            printf("%5s:%d\n",leader[i].name,leader[i].count);
      return 0;
}
```

定义结构体数组

(1) 定义结构体数组一般形式是

① **struct 结构体名**
{成员表列} 数组名[数组长度];

```
struct Person
{     char name[20];
      int count;
} leader[3];
```

② 先声明一个结构体类型，然后再用此类型定义结构体数组

结构体类型数组名[数组长度];

```
struct Person
{     char name[20];
      int count;
}
struct Person leader[3];    //leader是结构体数组名
```

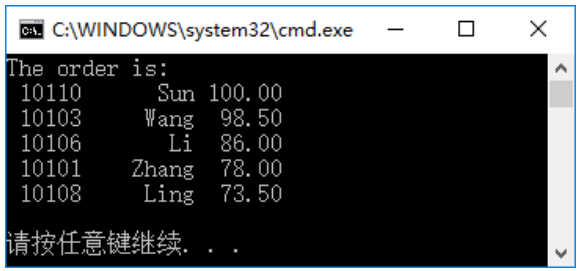
(2) 对结构体数组初始化的形式是在定义数组的后面加上：**={初值表列};**

```
struct Person leader[3]= {"Li",0,"Zhang",0,"Sun",0};
```

结构体数组的应用举例

【例9.4】有n个学生的信息(包括学号、姓名、成绩)，要求按照成绩的高低顺序输出各学生的信息。

```
#include <stdio.h>
struct Student                          //声明结构体类型struct Student
{     int num;
      char name[20];
      float score;
};
int main()
{     struct Student stu[5]={{10101,"Zhang",78},{10103,"Wang",98.5},{10106,"Li",86},
      {10108,"Ling",73.5},{10110,"Sun",100}};          //定义结构体数组并初始化
      struct Student temp;          //定义结构体变量temp，用作交换时的临时变量
      const int n=5;                //定义常变量n
      int i,j,k;
      printf("The order is:\n");
      for(i=0;i<n-1;i++)
      {     k=i;
            for(j=i+1;j<n;j++)
                  if(stu[j].score>stu[k].score)          //进行成绩的比较
                        k=j;
            temp=stu[k]; stu[k]=stu[i]; stu[i]=temp;     //stu[k]和stu[i]元素互换
      }
      for(i=0;i<n;i++)
            printf("%6d %8s %6.2f\n",stu[i].num,stu[i].name,stu[i].score);
      printf("\n");
      return 0;
}
```



结构体指针

结构体指针

所谓结构体指针就是指向结构体变量的指针，一个结构体变量的起始地址就是这个结构体变量的指针。如果把一个结构体变量的起始地址存放在一个指针变量中，那么，这个指针变量就指向该结构体变量。

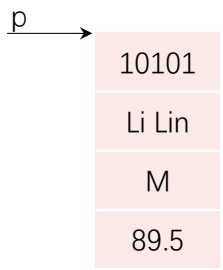
指向结构体变量的指针

【例9.5】通过指向结构体变量的指针变量输出结构体变量中成员的信息。

```
#include <stdio.h>
#include <string.h>
int main()
{    struct Student              //声明结构体类型struct Student
     {    long num;
          char name[20];
          char sex;
          float score;
     };
     struct Student stu_1;     //定义struct Student类型的变量stu_1
     struct Student *p;        //定义指向struct Student 类型数据的指针变量p
     p=&stu_1;                 //p指向stu_1
     stu_1.num=10101;          //对结构体变量的成员赋值
     strcpy(stu_1.name,"Li Lin");   //用字符串复制函数给stu_1.name赋值
     stu_1.sex='M';
     stu_1.score=89.5;
     printf("No.:%ld\nname:%s\nsex:%c\nscore:%5.1f\n",stu_1.num,stu_1.name,stu_1.sex,stu_1.score);   //输出结果
     printf("\nNo.:%ld\nname:%s\nsex:%c\nscore:%5.1f\n",(*p).num,(*p).name,(*p).sex, (*p).score);
     return 0;
}
```

```
C:\WINDOWS\system32\cmd.exe
No.:10101
name:Li Lin
sex:M
score: 89.5

No.:10101
name:Li Lin
sex:M
score: 89.5
请按任意键继续. . .
```

(*p).num也可表示为p->num



如果p指向一个结构体变量stu，以下3种用法等价：

① stu.成员名

stu.num

② (*p).成员名

(*p).num

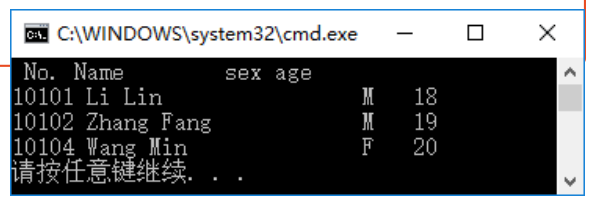
③ p->成员名

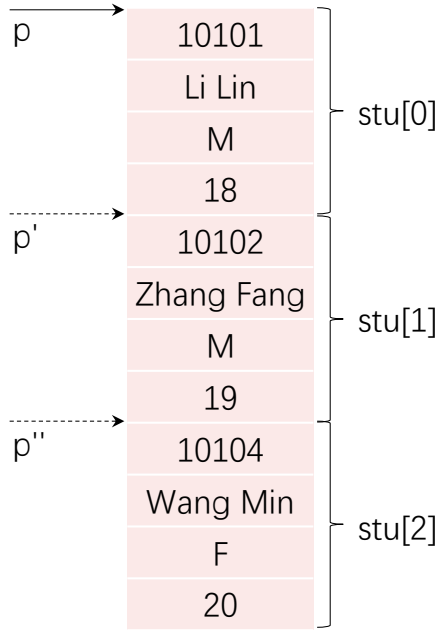
p->num

指向结构体数组的指针

【例9.6】有3个学生的信息，放在结构体数组中，要求输出全部学生的信息。

```
#include <stdio.h>
struct Student                    //声明结构体类型struct Student
{     int num;
      char name[20];
      char sex;
      int age;
};
struct Student stu[3]={{10101,"Li Lin",'M',18},{10102,"Zhang Fang",'M',19},{10104,"Wang Min",'F',20}};
//定义结构体数组并初始化
int main()
{     struct Student *p;          //定义指向struct Student结构体变量的指针变量
      printf(" No. Name       sex age\n");
      for (p=stu;p<stu+3;p++)
      printf("%5d %-20s %2c %4d\n",p->num, p->name, p->sex, p->age);      //输出结果
      return 0;
}
```





用结构体变量和结构体变量的指针作函数参数

将一个结构体变量的值传递给另一个函数，有3个方法：

(1) 用结构体变量的成员作参数。

例如，用stu[1].num或stu[2].name作函数实参，将实参值传给形参。用法和用普通变量作实参是一样的，属于“值传递”方式。应当注意实参与形参的类型保持一致。

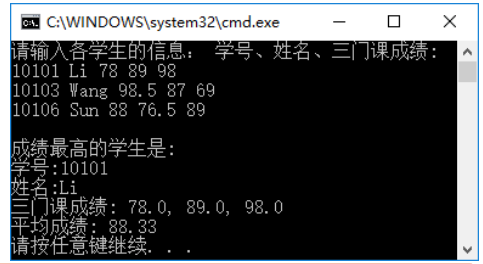
(2) 用结构体变量作实参。

用结构体变量作实参时，采取的也是“值传递”的方式，将结构体变量所占的内存单元的内容全部按顺序传递给形参，形参也必须是同类型的结构体变量。在函数调用期间形参也要占用内存单元。这种传递方式在空间和时间上开销较大，如果结构体的规模很大时，开销是很可观的。此外，由于采用值传递方式，如果在执行被调用函数期间改变了形参（也是结构体变量）的值，该值不能返回主调函数，这往往造成使用上的不便。因此一般较少用这种方法。

(3) 用指向结构体变量（或数组元素）的指针作实参，将结构体变量（或数组元素）的地址传给形参。

用结构体变量和结构体变量的指针作函数参数

【例9.7】有n个结构体变量，内含学生学号、姓名和3门课程的成绩。要求输出平均成绩最高的学生的信息(包括学号、姓名、3门课程成绩和平均成绩)。

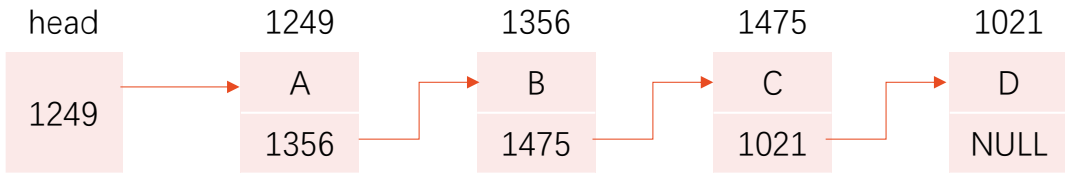



```
#include <stdio.h>
#define N 3                 //学生数为3
struct Student              //建立结构体类型struct Student
{     int num;              //学号
      char name[20];        //姓名
      float score[3];       //3门课成绩
      float aver;           //平均成绩
};
int main()
{     void input(struct Student stu[]);   //函数声明
      struct Student max(struct Student stu[]);    //函数声明
      void print(struct Student stu);     //函数声明
      struct Student stu[N],*p=stu;       //定义结构体数组和指针
      input(p);                           //调用input函数
      print(max(p)); //调用print函数,以max函数的返回值作为实参
      return 0;
}
void input(struct Student stu[])          //定义input函数
{     int i;
      printf("请输入各学生的信息：  学号、姓名、三门课成绩:\n");
      for(i=0;i<N;i++)
      {     scanf("%d %s %f %f %f",&stu[i].num,stu[i].name,
            &stu[i].score[0],&stu[i].score[1],&stu[i].score[2]); //输入数据
            stu[i].aver=(stu[i].score[0]+stu[i].score[1]+stu[i].score[2])/3.0;
            //求平均成绩
      }
}
struct Student max(struct Student stu[])    //定义max函数
{     int i,m=0;              //用m存放成绩最高的学生在数组中的序号
      for(i=0;i<N;i++)
      if(stu[i].aver>stu[m].aver) m=i;
      //找出平均成绩最高的学生在数组中的序号
      return stu[m];          //返回包含该生信息的结构体元素
}

void print(struct Student stud)           //定义print函数
{     printf("\n成绩最高的学生是:\n");
      printf("学号:%d\n姓名:%s\n三门课成绩:%5.1f,%5.1f,%5.1f\n平均成绩:
%6.2f\n",stud.num,stud.name,stud.score[0],stud.score[1],stud.score[2],st
ud.aver);
}
```

*用指针处理链表

什么是链表

链表是一种常见的重要的数据结构。它是动态地进行存储分配的一种结构。



链表有一个“**头指针**”变量，图中以head表示，它存放一个地址，该地址指向一个元素。

链表中每一个元素称为“**结点**”，每个结点都应包括两个部分：

(1) 用户需要用的实际数据；

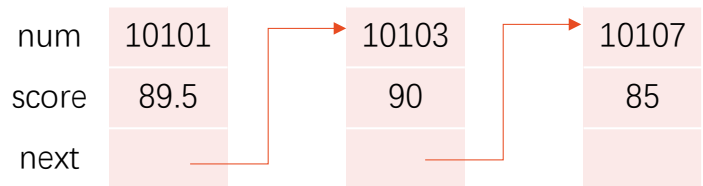
(2) 下一个结点的地址。

head指向第1个元素，第1个元素又指向第2个元素……直到最后一个元素，该元素不再指向其他元素，它称为“**表尾**”，它的地址部分放一个“**NULL**”（表示“空地址”），链表到此结束。

什么是链表

可以用结构体变量建立链表。一个结构体变量包含若干成员，这些成员可以是数值类型、字符类型、数组类型，也可以是指针类型。用指针类型成员来存放下一个结点的地址。

```
struct Student
{    int num;
     float score;
     struct Student *next;      //next是指针变量，指向结构体变量
};
```



成员num和score用来存放结点中的有用数据（用户需要用到的数据）。

next是指针类型的成员，它指向struct Student类型数据（就是next所在的结构体类型）

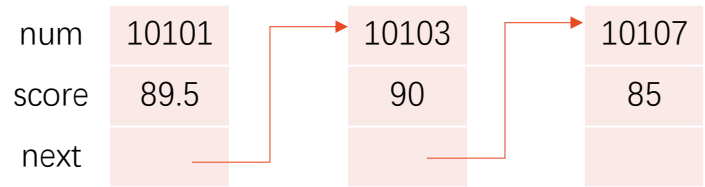
注意

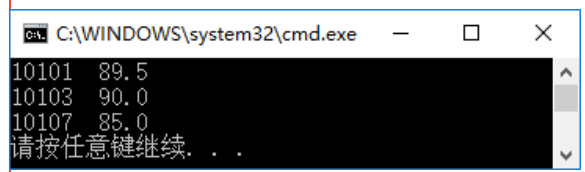
- 上面只是定义了一个struct Student类型，并未实际分配存储空间，只有定义了变量才分配存储单元。

建立简单的静态链表

【例9.8】建立一个简单链表，它由3个学生数据的结点组成，要求输出各结点中的数据。

```
#include <stdio.h>
struct Student                          //声明结构体类型struct Student
{       int num;
        float score;
        struct Student*next;
};
int main()
{       struct Student a,b,c,*head,*p;      //定义3个结构体变量a,b,c作为链表的结点
        a.num=10101; a.score=89.5;          //对结点a的num和score成员赋值
        b.num=10103; b.score=90;            //对结点b的num和score成员赋值
        c.num=10107; c.score=85;            //对结点c的num和score成员赋值
        head=&a;                            //将结点a的起始地址赋给头指针head
        a.next=&b;                          //将结点b的起始地址赋给a结点的next成员
        b.next=&c;                          //将结点c的起始地址赋给a结点的next成员
        c.next=NULL;                        //c结点的next成员不存放其他结点地址
        p=head;                             //使p指向a结点
        do
        {       printf("%ld %5.1f\n",p->num,p->score);      //输出p指向的结点的数据
                p=p->next;                  //使p指向下一结点
        }while(p!=NULL);                    //输出完c结点后p的值为NULL，循环终止
        return 0;
}
```

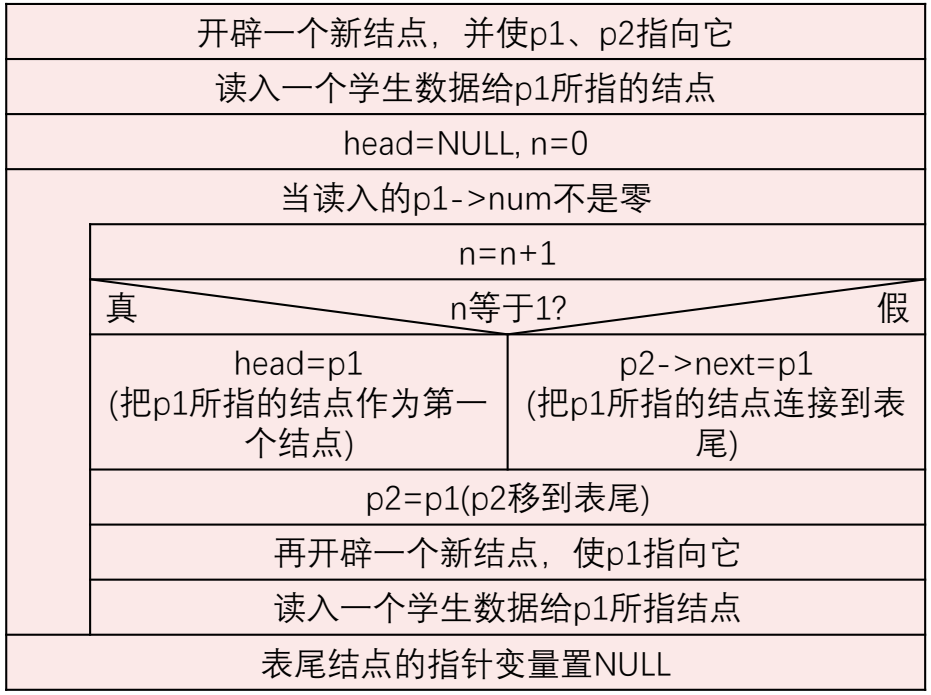




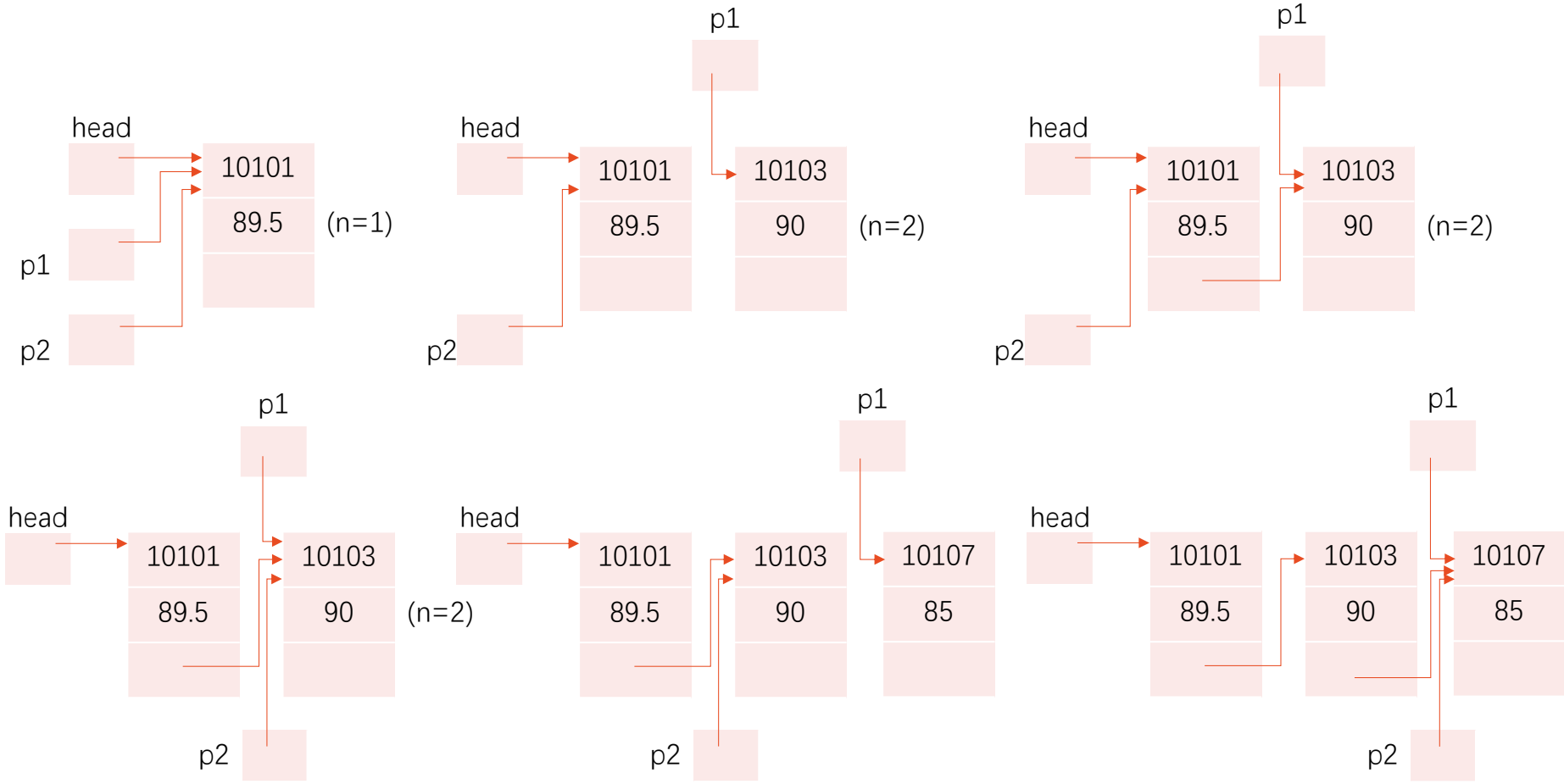
建立简单的动态链表

【例9.9】写一函数建立一个有3名学生数据的单向动态链表。

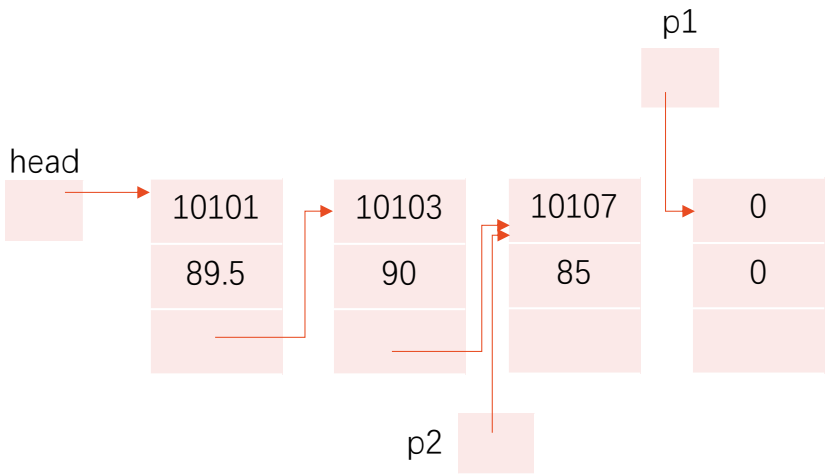
所谓建立动态链表是指在程序执行过程中从无到有地建立起一个链表，即一个一个地开辟结点和输入各结点数据，并建立起前后相链的关系。

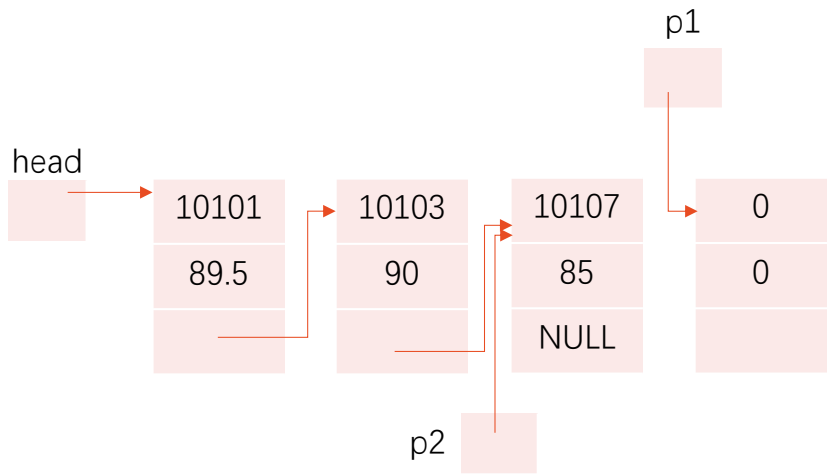


建立简单的动态链表【例9.9】写一函数建立一个有3名学生数据的单向动态链表。



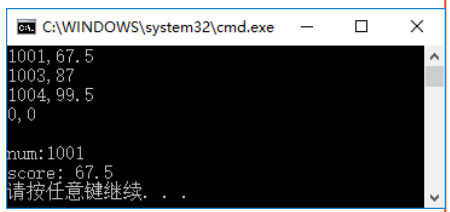
建立简单的动态链表【例9.9】写一函数建立一个有3名学生数据的单向动态链表。





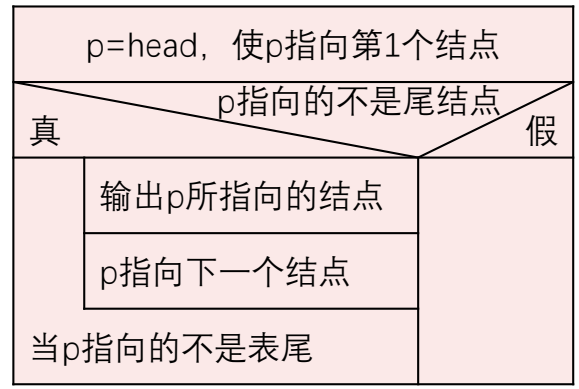
建立简单的动态链表【例9.9】写一函数建立一个有3名学生数据的单向动态链表。

```
#include <stdio.h>
#include <stdlib.h>
#define LEN sizeof(struct Student)
struct Student
{     long num;
      float score;
      struct Student*next;
};
int n;      //n为全局变量，本文件模块中各函数均可使用它
struct Student *creat(void)
//定义函数。此函数返回一个指向链表头的指针
{     struct Student *head;
      struct Student *p1,*p2;
      n=0;
      p1=p2=(struct Student*) malloc(LEN);  //开辟一个新单元
      scanf("%ld,%f",&p1->num,&p1->score);
      //输入第1个学生的学号和成绩
      head=NULL;
      while(p1->num!=0)
      {     n=n+1;
            if(n==1) head=p1;
            else p2->next=p1;
            p2=p1;
            p1=(struct Student*)malloc(LEN);
            //开辟动态存储区，把起始地址赋给p1
            scanf("%ld,%f",&p1->num,&p1->score);
            //输入其他学生的学号和成绩
      }
      p2->next=NULL;
      return(head);
}
int main()
{     struct Student *pt;
      pt=creat();     //函数返回链表第一个结点的地址
      printf("\nnum:%ld\nscore:%5.1f\n",pt->num,pt->score);
      //输出第1个结点的成员值
      return 0;
};
```



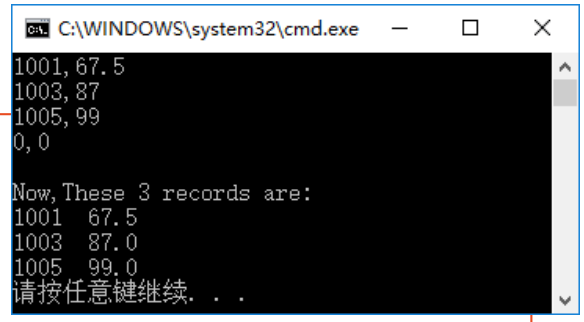
输出链表【例9.10】编写一个输出链表的函数print。

```
#include <stdio.h>
#include <stdlib.h>
#define LEN sizeof(struct Student)
struct Student                          //声明结构体类型struct Student
{     long num;
      float score;
      struct Student *next;
};
int n;                                  //全局变量n
void print(struct Student*head)         //定义print函数
{     struct Student*p;                 //在函数中定义struct Student类型的变量p
      printf("\nNow,These %d records are:\n",n);
      p=head;                           //使p指向第1个结点
      if(head!=NULL)                    //若不是空表
            do
            {     printf("%ld %5.1f\n",p->num,p->score);      //输出一个结点中的学号与成绩
                  p=p->next;            //p指向下一个结点
            }while(p!=NULL);            //当p不是"空地址"
}
```



组合【例9.7】和【例9.9】

```
#include <stdio.h>
#include <malloc.h>
#define LEN sizeof(struct Student)
struct Student
{     long num;
      float score;
      struct Student *next;
};
int n;
struct Student *creat()  //建立链表的函数
{     struct Student *head;
      struct Student *p1,*p2;
      n=0;
      p1=p2=(struct Student *)malloc(LEN);
      scanf("%ld,%f",&p1->num,&p1->score);
      head=NULL;
      while(p1->num!=0)
      {     n=n+1;
            if(n==1) head=p1;
            else p2->next=p1;
            p2=p1;
            p1=(struct Student *)malloc(LEN);
            scanf("%ld,%f",&p1->num,&p1->score);
      }
      p2->next=NULL;
      return(head);
}

void print(struct Student *head)  //输出链表的函数
{     struct Student *p;
      printf("\nNow,These %d records are:\n",n);
      p=head;
      if(head!=NULL)
            do
            {     printf("%ld %5.1f\n",p->num,p->score);
                  p=p->next;
            }while(p!=NULL);
}

int main()
{     struct Student *head;
      head=creat(); //调用creat函数，返回第1个结点的起始地址
      print(head);   //调用print函数
      return 0;
}
```



*共用体类型

什么是共用体类型

使几个不同类型的变量共享同一段内存的结构，称为 **“共用体”类型**的结构。

union共用体名
{　　成员表列
}变量表列;


```
union Data
{     int i;
      //表示不同类型的变量i,ch,f可以存放到同一段存储单元中
      char ch;
      float f;
}a,b,c;                    //在声明类型同时定义变量
```

```
union Data              //声明共用体类型
{     int i;
      char ch;
      float f;
};
union Data a,b,c;       //用共用体类型定义变量
```

```
union          //没有定义共用体类型名
{     int i;
      char ch;
      float f;
}a,b,c;
```

“共用体”与“结构体”的定义形式相似。

但它们的含义是不同的。

结构体变量所占内存长度是各成员占的内存长度之和。每个成员分别占有其自己的内存单元。而共用体变量所占的内存长度等于最长的成员的长度。几个成员共用一个内存区。

引用共用体变量的方式

只有先定义了共用体变量才能引用它，但应注意，不能引用共用体变量，而只能引用共用体变量中的成员。

```
a.i      //引用共用体变量中的整型变量i
a.ch     //引用共用体变量中的字符变量ch
a.f      //引用共用体变量中的实型变量f
```

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

```
printf("%d",a.i);
```

共用体类型数据的特点

(1) 同一个内存段可以用来存放几种不同类型的成员，但在每一瞬时只能存放其中一个成员，而不是同时存放几个。

```
union Date
{     int i;
      char ch;
      float f;
}a;
a.i=97;
printf("%d",a.i);    //输出整数97
printf("%c",a.ch); //输出字符'a'
printf("%f",a.f);    //输出实数0.000000
```

(2) 可以对共用体变量初始化，但初始化表中只能有一个常量。

```
union Data a={1,'a',1.5};
```

(3) 共用体变量中起作用的成员是最后一次被赋值的成员，在对共用体变量中的一个成员赋值后，原有变量存储单元中的值就被取代。

(4) 共用体变量的地址和它的各成员的地址都是同一地址。

(5) 不能对共用体变量名赋值，也不能企图引用变量名来得到一个值。C 99允许同类型的共用体变量互相赋值。

```
 a=1; //不能对共用体变量赋值，赋给谁?
m=a; //企图引用共用体变量名以得到一个值赋给整型变量m
```

(6) C 99允许用共用体变量作为函数参数。

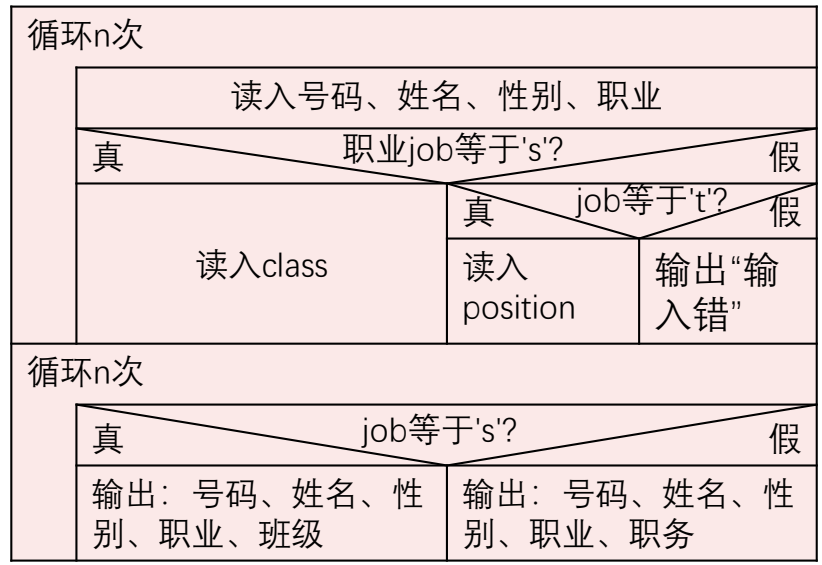
```
b=a;     //a和b是同类型的共用体变量，合法
```

(7) 共用体类型可以出现在结构体类型定义中，也可以定义共用体数组。反之，结构体也可以出现在共用体类型定义中，数组也可以作为共用体的成员。

共用体类型数据的特点

【例9.10】有若干个人员的数据，其中有学生和教师。学生的数据中包括： 姓名、号码、性别、职业、班级。教师的数据包括： 姓名、号码、性别、职业、职务。要求用同一个表格来处理。

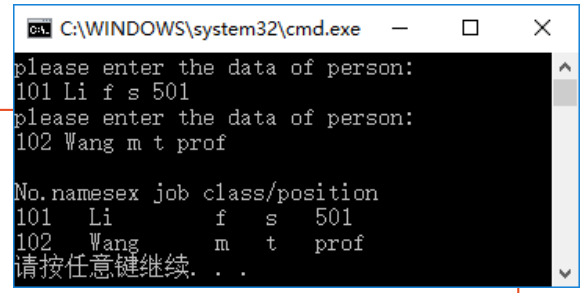
num	name	sex	job	class(班) position(职务)
101	Li	f	s	501
102	Wang	m	t	prof



共用体类型数据的特点

【例9.10】有若干个人员的数据，其中有学生和教师。学生的数据中包括： 姓名、号码、性别、职业、班级。教师的数据包括： 姓名、号码、性别、职业、职务。要求用同一个表格来处理。

```
#include <stdio.h>
struct                          //声明无名结构体类型
{     int num;                  //成员num(编号)
      char name[10];            //成员name(姓名)
      char sex;                 //成员sex(性别)
      char job;                 //成员job(职业)
      union                     //声明无名共用体类型
      {     int clas;           //成员clas(班级)
            char position[10];  //成员position(职务)
      }category;                //成员category是共用体变量
}person[2];                     //定义结构体数组person，有两个元素
```




```
int main()
{   int i;
    for(i=0;i<2;i++)
    {   printf("please enter the data of person:\n");
        scanf("%d %s %c %c",&person[i].num,person[i].name,&person[i].sex,&person[i].job);      //输入前4项
        if(person[i].job=='s')
            scanf("%d",&person[i].category.clas);                                            //如是学生，输入班级
        else if(person[i].job=='t')
            scanf("%s",person[i].category.position);                                         //如是教师，输入职务
        else
            printf("Input error!");                                                         //如job不是's'和't'，显示“输入错误”
    }
    printf("\n");
    printf("No.namesex job class/position\n");
    for(i=0;i<2;i++)
    {   if (person[i].job=='s')                                                             //若是学生
            printf("%-6d%-10s%-4c%-4c%-10d\n",person[i].num,person[i].name,person[i].sex,person[i].job,person[i].category.clas);
        else                                                                                //若是教师
            printf("%-6d%-10s%-4c%-4c%-10s\n",person[i].num, person[i].name,person[i].sex,person[i].job,person[i].category.position);
    }
    return 0;
}
```

使用枚举类型

使用枚举类型

enum [枚举名]{枚举元素列表}

enum Weekday{sun，mon，tue，wed，thu，fri，sat};

如果一个变量只有几种可能的值，则可以定义为**枚举**(enumeration)**类型**，所谓“枚举”就是指把可能的值一一列举出来，变量的值只限于列举出来的值的范围内。

声明枚举类型用enum开头。花括号中的sun,mon,…,sat称为**枚举元素**或**枚举常量**。

enum Weekday workday,weekend;

枚举类型　　枚举变量

也可以不声明有名字的枚举类型，而直接定义枚举变量：

enum {sun，mon，tue，wed，thu，fri，sat}workday,weekend;

(1) C编译对枚举类型的枚举元素按常量处理，故称枚举常量。不要因为它们是标识符(有名字)而把它们看作变量，不能对它们赋值。

(2) 每一个枚举元素都代表一个整数，C语言编译按定义时的顺序默认它们的值为0,1,2,3,4,5…。也可以在定义枚举类型时显式地指定枚举元素的数值。

(3) 枚举元素可以用来作判断比较。枚举元素的比较规则是按其在初始化时指定的整数来进行比较的。

使用枚举类型

【例9.12】口袋中有红、黄、蓝、白、黑5种颜色的球若干个。每次从口袋中先后取出3个球，问得到3种不同颜色的球的可能取法，输出每种排列的情况。

```
#include <stdio.h>
int main()
{       enum Color {red,yellow,blue,white,black};                       //声明枚举类型enum Co
        enum Color i,j,k,pri;                                           //定义枚举变量i,j,k,pri
        int n,loop;
        n=0;
        for(i=red;i<=black;i++)                                         //外循环使i的值从red变
                for(j=red;j<=black;j++)                                 //中循环使j的值从red变
                        if(i!=j)//如果二球不同色
                        {       for (k=red;k<=black;k++)                //内循环使k的值从red变
                                        if ((k!=i) && (k!=j))           //如果3球不同色
                                        {       n=n+1;                  //符合条件的次数加1
                                                printf("%-4d",n);       //输出当前是第几个符合
                                                for(loop=1;loop<=3;loop++)      //先后对3个球分别处理
                                                {       switch (loop)           //loop的值从1变到3
                                                        {       case 1: pri=i;break;    //loop的值为1时，把第
                                                                case 2: pri=j;break;    //loop的值为2时，把第
                                                                case 3: pri=k;break;    //loop的值为3时，把第
                                                                default:break;
                                                        }
                                                        switch (pri)//根据球的颜色输出相应的文字
                                                        {       case red:printf("%-10s","red");break;           //pri的
                                                                case yellow: printf("%-10s","yellow");break;    //pri的
                                                                case blue: printf("%-10s","blue");break;        //pri的
                                                                case white: printf("%-10s","white");break;      //pri的
                                                                case black: printf("%-10s","black"); break;     //pri的
                                                                default:break;
                                                        }
                                                }
                                                printf("\n");
                                        }
                        }
        printf("\ntotal:%5d\n",n);
        return 0;
}
```

```
C:\Users\jin\Desktop\未命名1.exe
1   red     yellow  blue
2   red     yellow  white
3   red     yellow  black
4   red     blue    yellow
5   red     blue    white
6   red     blue    black
7   red     white   yellow
8   red     white   blue
9   red     white   black
10  red     black   yellow
11  red     black   blue
12  red     black   white
13  yellow  red     blue
14  yellow  red     white
15  yellow  red     black
16  yellow  blue    red
17  yellow  blue    white
18  yellow  blue    black
19  yellow  white   red
20  yellow  white   blue
21  yellow  white   black
22  yellow  black   red
23  yellow  black   blue
24  yellow  black   white
25  blue    red     yellow
26  blue    red     white
27  blue    red     black
28  blue    yellow  red
29  blue    yellow  white
30  blue    yellow  black
31  blue    white   red
32  blue    white   yellow
33  blue    white   black
34  blue    black   red
35  blue    black   yellow
36  blue    black   white
37  white   red     yellow
38  white   red     blue
39  white   red     black
40  white   yellow  red
41  white   yellow  blue
42  white   yellow  black
43  white   blue    red
44  white   blue    yellow
45  white   blue    black
46  white   black   red
47  white   black   yellow
48  white   black   blue
49  black   red     yellow
50  black   red     blue
51  black   red     white
52  black   yellow  red
53  black   yellow  blue
54  black   yellow  white
55  black   blue    red
56  black   blue    yellow
57  black   blue    white
58  black   white   red
59  black   white   yellow
60  black   white   blue

total:   60
```

*用typedef声明新类型名

用typedef声明新类型名

1. 简单地用一个新的类型名代替原有的类型名

```
typedef int Integer;   //指定用Integer为类型名，作用与int相同
typedef float Real;    //指定用Real为类型名，作用与float相同
```

2. 命名一个简单的类型名代替复杂的类型表示方法

① 命名一个新的类型名代表结构体类型　　② 命名一个新的类型名代表数组类型

③ 命名一个新的类型名代表指针类型　　④命名一个新的类型名代表指向函数的指针类型

```
typedef struct
{     int month;
      int day;
      int year;
}Date;                  //声明了一个新类型名Date，代表结构体类型
Date birthday;          //定义结构体类型变量birthday，不要写成struct Date birthday;   ①
Date*p;                 //定义结构体指针变量p，指向此结构体类型数据

typedef int Num[100];   //声明Num为整型数组类型名                                    ②
Num a;                  //定义a为整型数组名，它有100个元素

typedef char*String;    //声明String为字符指针类型                                   ③
String p,s[10];         //定义p为字符指针变量，s为字符指针数组

ypedef int (*Pointer)(); //声明Pointer为指向函数的指针类型，该函数返回整型值        ④
Pointer p1,p2;          //p1,p2为Pointer类型的指针变量
```

用typedef声明新类型名

声明一个新的类型名的方法是：

① 先按定义变量的方法写出定义体（如： int i;）　② 将变量名换成新类型名（例如： 将i换成Count）

③ 在最前面加typedef（例如： typedef int Count）　④ 然后可以用新类型名去定义变量

简单地说，就是按定义变量的方式把变量名换上新类型名，并在最前面加typedef，就声明了新类型名代表原来的类型。

以定义上述的数组类型为例来说明：

① 先按定义数组变量形式书写： int a[100]　② 将变量名a换成自己命名的类型名： int Num[100]

③ 在前面加上typedef，得到typedef int Num[100]　④ 用来定义变量： Num a; 相当于定义了： int a[100];

同样，对字符指针类型，也是：

① char *p;　//定义变量p的方式　② char *String;　//用新类型名String取代变量名p

③ typedef char *String;　//加typedef　④ String p;　//用新类型名String定义变量，相当char *p;

习惯上，常把用typedef声明的类型名的第1个字母用大写表示，以便与系统提供的标准类型标识符相区别。

用typedef声明新类型名

(1) typedef的方法实际上是为特定的类型指定了一个同义字(synonyms)。

(2) 用typedef只是对已经存在的类型指定一个新的类型名，而没有创造新的类型。

(3) 用tyoedef声明数组类型、指针类型，结构体类型、共用体类型、枚举类型等，使得编程更加方便。

(4) typedef与#define表面实质不同的。#define是在预编译时处理的，它只能作简单的字符串替换，而typedef是在编译阶段处理的，且并非简单的字符串替换。

(5) 当不同源文件中用到同一类型数据（尤其是像数组、指针、结构体、共用体等类型数据）时，常用typedef声明一些数据类型。可以把所有的typedef名称声明单独放在一个头文件中，然后在需要用到它们的文件中用#include指令把它们包含到文件中。这样编程者就不需要在各文件中自己定义typedef名称了。

(6) 使用typedef名称有利于程序的通用与移植。有时程序会依赖于硬件特性，用typedef类型就便于移植。

第10章

对文件的输入输出

C文件的有关基本知识

什么是文件

文件有不同的类型，在程序设计中，主要用到两种文件:

(1) **程序文件**。包括源程序文件(后缀为.c)、目标文件(后缀为.obj)、可执行文件(后缀为.exe)等。这种文件的内容是程序代码。

(2) **数据文件**。文件的内容不是程序，而是供程序运行时读写的数据，如在程序运行过程中输出到磁盘(或其他外部设备)的数据，或在程序运行过程中供读入的数据。如一批学生的成绩数据、货物交易的数据等。

为了简化用户对输入输出设备的操作，使用户不必去区分各种输入输出设备之间的区别，**操作系统把各种设备都统一作为文件来处理**。从操作系统的角度看，每一个与主机相连的输入输出设备都看作一个文件。例如，终端键盘是输入文件，显示屏和打印机是输出文件。

什么是文件

文件（file）一般指**存储在外部介质上数据的集合**。操作系统是以文件为单位对数据进行管理的。

输入输出是数据传送的过程，数据如流水一样从一处流向另一处，因此常将输入输出形象地称为**流**(stream)，即**数据流**。流表示了信息从源到目的端的流动。在输入操作时，数据从文件流向计算机内存，在输出操作时，数据从计算机流向文件(如打印机、磁盘文件)。

C语言把文件看作一个字符(或字节)的序列，即由一个一个字符（或字节）的数据顺序组成。一个输入输出流就是一个字符流或字节(内容为二进制数据)流。

C的数据文件由一连串的字符（或字节）组成，而不考虑行的界限，两行数据间不会自动加分隔符，对文件的存取是以字符（字节）为单位的。输入输出数据流的开始和结束仅受程序控制而不受物理符号（如回车换行符）控制，这就增加了处理的灵活性。这种文件称为流式文件。

文件名

一个文件要有一个唯一的文件标识，以便用户识别和引用。

文件标识包括3部分：(1)文件路径；(2)文件名主干；(3)文件后缀。

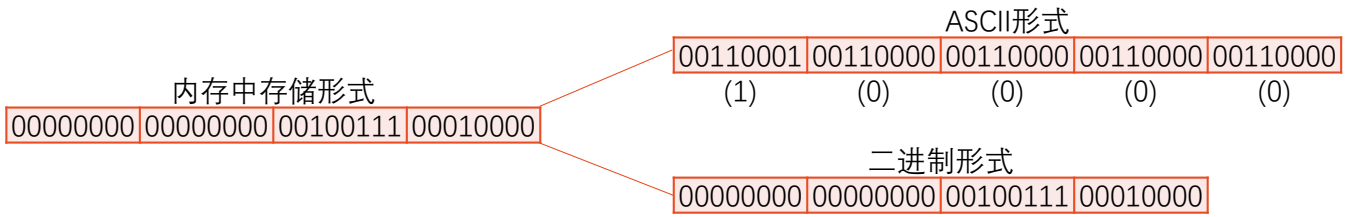
- 文件路径表示文件在外部存储设备中的位置。
- 文件名主干的命名规则遵循标识符的命名规则。
- 文件后缀用来表示文件的性质。

为方便起见，文件标识常被称为文件名，但应了解此时所称的文件名，实际上包括以上3部分内容，而不仅是文件名主干。

文件的分类

根据数据的组织形式，数据文件可分为**ASCII文件**和**二进制文件**。数据在内存中是以二进制形式存储的，如果不加转换地输出到外存，就是二进制文件，可以认为它就是存储在内存的数据的映像，所以也称之为**映像文件**(image file)。如果要求在外存上以ASCII代码形式存储，则需要在存储前进行转换。ASCII文件又称**文本文件**（text file），每一个字节存放一个字符的ASCII代码。

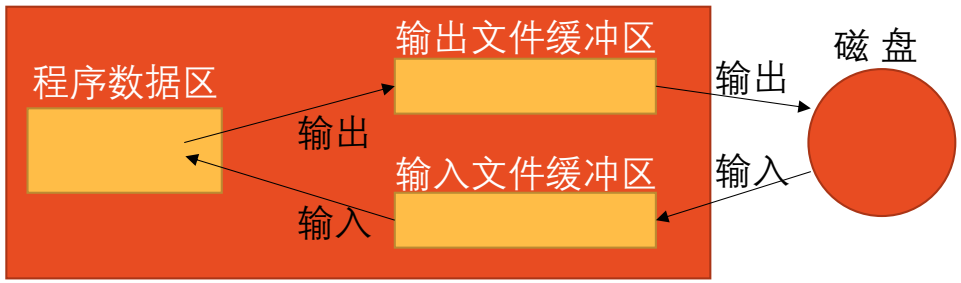
字符一律以ASCII形式存储，数值型数据既可以用ASCII形式存储，也可以用二进制形式存储。



用ASCII码形式输出时字节与字符一一对应，一个字节代表一个字符，因而便于对字符进行逐个处理，也便于输出字符。但一般占存储空间较多，而且要花费转换时间（二进制形式与ASCII码间的转换）。用二进制形式输出数值，可以节省外存空间和转换时间，把内存中的存储单元中的内容原封不动地输出到磁盘(或其他外部介质)上，此时每一个字节并不一定代表一个字符。

文件缓冲区

ANSI C标准采用“**缓冲文件系统**”处理数据文件，所谓缓冲文件系统是指系统自动地在内存区为程序中每一个正在使用的文件开辟一个文件缓冲区。从内存向磁盘输出数据必须先送到内存中的缓冲区，装满缓冲区后才一起送到磁盘去。如果从磁盘向计算机读入数据，则一次从磁盘文件将一批数据输入到内存缓冲区（充满缓冲区），然后再从缓冲区逐个地将数据送到程序数据区（给程序变量）。这样做是为了节省存取时间，提高效率，缓冲区的大小由各个具体的C编译系统确定。



说明：每一个文件在内存中只有一个缓冲区，在向文件输出数据时，它就作为输出缓冲区，在从文件输入数据时，它就作为输入缓冲区。

文件类型指针

缓冲文件系统中，关键的概念是“**文件类型指针**”，简称“**文件指针**”。每个被使用的文件都在内存中开辟一个相应的文件信息区，用来存放文件的有关信息（如文件的名字、文件状态及文件当前位置等）。这些信息是保存在一个结构体变量中的。该结构体类型是由系统声明的，取名为**FILE**。

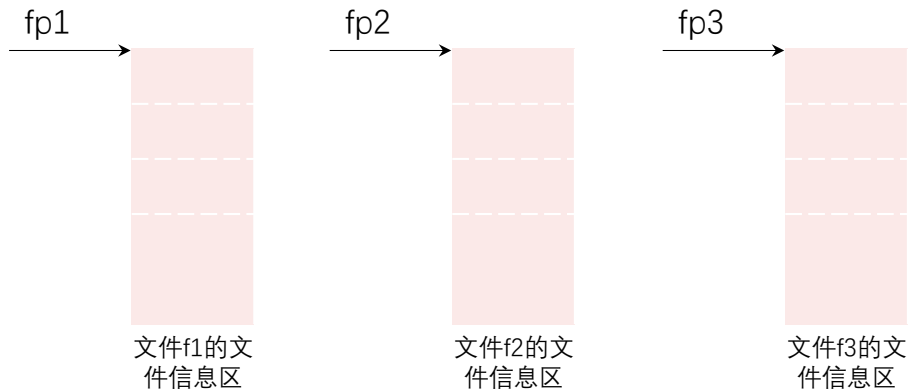
```
typedef struct
{    short level;              //缓冲区“满”或“空”的程度
     unsigned flags;           //文件状态标志
     char fd;                  //文件描述符
     unsigned char hold;       //如缓冲区无内容不读取字符
     short bsize;              //缓冲区的大小
     unsigned char*buffer;     //数据缓冲区的位置
     unsigned char*curp;       //文件位置标记指针当前的指向
     unsigned istemp;          //临时文件指示器
     short token;              //用于有效性检查
}FILE;
```

一种C编译环境提供的stdio.h头文件中有以下的文件类型声明

文件类型指针

```
FILE *fp;
//定义一个指向FILE类型数据的指针变量
```

可以使fp指向某一个文件的文件信息区(是一个结构体变量)，通过该文件信息区中的信息就能够访问该文件。也就是说，**通过文件指针变量能够找到与它关联的文件**。如果有n个文件，应设n个指针变量，分别指向n个FILE类型变量，以实现对n个文件的访问。为方便起见，通常将这种指向文件信息区的指针变量简称为**指向文件的指针变量**。



注意

- 指向文件的指针变量并不是指向外部介质上的数据文件的开头，而是指向内存中的文件信息区的开头。

打开与关闭文件

打开与关闭文件

对文件读写之前应该“打开”该文件，在使用结束之后应“关闭”该文件。

所谓“打开”是指为文件建立相应的信息区(用来存放有关文件的信息)和文件缓冲区(用来暂时存放输入输出的数据)。

在编写程序时，在打开文件的同时，一般都指定一个指针变量指向该文件，也就是建立起指针变量与文件之间的联系，这样，就可以通过该指针变量对文件进行读写了。

所谓“关闭”是指撤销文件信息区和文件缓冲区，使文件指针变量不再指向该文件，显然就无法进行对文件的读写了。

用fopen函数打开数据文件

fopen(文件名，使用文件方式)；

```
FILE*fp;                  //定义一个指向文件的指针变量fp
fp=fopen("a1","r");       //将fopen函数的返回值赋给指针变量fp
```

表示以“读入”方式打开名字为a1的文件

在打开一个文件时，通知编译系统以下3个信息：

① 需要打开文件的名字，也就是准备访问的文件的名字

② 使用文件的方式（“读”还是“写”等）

③ 让哪一个指针变量指向被打开的文件

用fopen函数打开数据文件

fopen(文件名，使用文件方式)；

使用文件方式

文件使用方式	含义	如果指定的文件不存在
“r”（只读）	为了输入数据，打开一个已存在的文本文件	出错
“w”（只写）	为了输出数据，打开一个文本文件	建立新文件
“a”（追加）	向文本文件尾添加数据	出错
“rb”（只读）	为了输入数据，打开一个二进制文件	出错
“wb”（只写）	为了输出数据，打开一个二进制文件	建立新文件
“ab”（追加）	向二进制文件尾添加数据	出错
“r+”（读写）	为了读和写，打开一个文本文件	出错
“w+”（读写）	为了读和写，建立一个新的文本文件	建立新文件
“a+”(读写）	为了读和写，打开一个文本文件	出错
“rb+”（读写）	为了读和写，打开一个二进制文件	出错
“wb+”（读写）	为了读和写，建立一个新的二进制文件	建立新文件
“ab+”（读写）	为读写打开一个二进制文件	出错

用fopen函数打开数据文件

fopen(文件名，使用文件方式)；

(1) 用“r”方式打开的文件只能用于向计算机输入而不能用作向该文件输出数据，而且该文件应该已经存在，并存有数据，这样程序才能从文件中读数据。不能用“r”方式打开一个并不存在的文件，否则出错。

(2) 用“w”方式打开的文件只能用于向该文件写数据（即输出文件），而不能用来向计算机输入。如果原来不存在该文件，则在打开文件前新建立一个以指定的名字命名的文件。如果原来已存在一个以该文件名命名的文件，则在打开文件前先将该文件删去，然后重新建立一个新文件。

(3) 如果希望向文件末尾添加新的数据（不希望删除原有数据），则应该用“a”方式打开。但此时应保证该文件已存在；否则将得到出错信息。在每个数据文件中自动设置了一个隐式的“**文件读写位置标记**”，它指向的位置就是当前进行读写的位置。如果“文件读写位置标记”在文件开头，则下一次的读写就是文件开头的数据。然后“文件读写位置标记”自动移到下一个读写位置，以便读写下一个数据。以添加方式打开文件时，文件读写位置标记移到文件末尾。

(4) 用“r+”“w+”“a+”方式打开的文件既可用来输入数据，也可用来输出数据。

用fopen函数打开数据文件

fopen(文件名，使用文件方式)；

(5) 如果不能实现“打开”的任务，fopen函数将会带回一个空指针值NULL。

(6) C标准建议用表10.1列出的文件使用方式打开文本文件或二进制文件，但目前使用的有些C编译系统可能不完全提供所有这些功能，需要注意所用系统的规定。

```
if ((fp=fopen("file1","r"))==NULL)
{    printf("cannot open this file\n");
     exit(0);
}
```

打开一个文件的常用方法

(7) 有12种文件使用方式，其中有6种是在第一个字母后面加了字母b的(如rb,wb,ab,rb+,wb+,ab+)，b表示二进制方式。其实，带b和不带b只有一个区别，即对换行的处理。由于在C语言用一个'\n'即可实现换行，而在Windows系统中为实现换行必须要用“回车”和“换行”两个字符，即'\r'和'\n'。因此，如果使用的是文本文件并且用“w”方式打开，在向文件输出时，遇到换行符'\n'时，系统就把它转换为'\r'和'\n'两个字符，否则在Windows系统中查看文件时，各行连成一片，无法阅读。同样，如果有文本文件且用“r”方式打开，从文件读入时，遇到'\r'和'\n'两个连续的字符，就把它们转换为'\n'一个字符。如果使用的是二进制文件，在向文件读写时，不需要这种转换。加b表示使用的是二进制文件，系统就不进行转换。

用fopen函数打开数据文件

fopen(文件名，使用文件方式)；

(8) 如果用“wb”的文件使用方式，并不意味着在文件输出时把内存中按ASCII形式保存的数据自动转换成二进制形式存储。输出的数据形式是由程序中采用什么读写语句决定的。例如，用fscanf和fprintf函数是按ASCII方式进行输入输出，而fread和fwrite函数是按二进制进行输入输出。

(9) 程序中可以使用3个标准的流文件——标准输入流、标准输出流和标准出错输出流。系统已对这3个文件指定了与终端的对应关系。标准输入流是从终端的输入，标准输出流是向终端的输出，标准出错输出流是当程序出错时将出错信息发送到终端。程序开始运行时系统自动打开这3个标准流文件。

用fclose函数关闭数据文件

fclose(文件指针)；；

```
fclose(fp);
```

在使用完一个文件后应该关闭它，以防止它再被误用。“关闭”就是撤销文件信息区和文件缓冲区，使文件指针变量不再指向该文件，也就是文件指针变量与文件“脱钩”，此后不能再通过该指针对原来与其相联系的文件进行读写操作，除非再次打开，使该指针变量重新指向该文件。

如果不关闭文件就结束程序运行将会丢失数据。因为，在向文件写数据时，是先将数据输出到缓冲区，待缓冲区充满后才正式输出给文件。如果当数据未充满缓冲区时程序结束运行，就有可能使缓冲区中的数据丢失。用fclose函数关闭文件时，先把缓冲区中的数据输出到磁盘文件，然后才撤销文件信息区。有的编译系统在程序结束前会自动先将缓冲区中的数据写到文件，从而避免了这个问题，但还是应当养成在程序终止之前关闭所有文件的习惯。

fclose函数也带回一个值，当成功地执行了关闭操作，则返回值为0；否则返回EOF(-1)。

顺序读写数据文件

怎样向文件读写字符

读写一个字符的函数:

函数名	调用形式	功能	返回值
fgetc	fgetc(fp)	从fp指向的文件读入一个字符	读成功，带回所读的字符，失败则返回文件结束标志EOF(即-1)
fputc	fputc(ch,fp)	把字符ch写到文件指针变量fp所指向的文件中	输出成功，返回值就是输出的字符；输出失败，则返回EOF（即-1）

fgetc的第1个字母f代表文件(file)，中间的get表示“获取”，最后一个字母c表示字符(character)，fgetc的含义很清楚：从文件读取一个字符。fputc也类似。

怎样向文件读写字符

【例10.1】从键盘输入一些字符，并逐个把它们送到磁盘上去，直到用户输入一个“#”为止。

用来存储数据的文件名可以在fopen函数中直接写成字符串常量形式，也可以在程序运行时由用户临时指定。

用fopen函数打开一个“只写”的文件(“w”表示只能写入不能从中读数据)，若成功，函数返回该文件所建立的信息区的起始地址给文件指针变量fp。若失败，则显示“无法打开此文件”，用exit函数终止程序运行，此函数在stdlib.h头文件中。

用getchar函数接收用户从键盘输入的字符。注意每次只能接收一个字符。

```
#include <stdio.h>
#include <stdlib.h>
int main()
{   FILE *fp;                                   //定义文件指针fp
    char ch,filename[10];
    printf("请输入所用的文件名: ");
    scanf("%s",filename);                       //输入文件名
    getchar();                                  //用来消化最后输入的回车符
    if((fp=fopen(filename,"w"))==NULL)          //打开输出文件并使fp指向此文件
    {   printf("cannot open file\n");           //如果打开出错就输出“打不开”
        exit(0);                                //终止程序
    }
    printf("请输入一个准备存储到磁盘的字符串(以#结束): ");
    ch=getchar();                               //接收从键盘输入的第一个字符
    while(ch!='#')                              //当输入'#'时结束循环
    {   fputc(ch,fp);                           //向磁盘文件输出一个字符
        putchar(ch);                            //将输出的字符显示在屏幕上
        ch=getchar();                           //再接收从键盘输入的一个字符
    }
    fclose(fp);                                 //关闭文件
    putchar(10);                                //向屏幕输出一个换行符
    return 0;
}
```

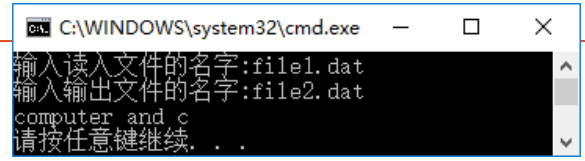
```
C:\WINDOWS\system32\cmd.exe
请输入所用的文件名: file1.dat
请输入一个准备存储到磁盘的字符串(以#结束): computer and c#
computer and c
请按任意键继续. . .
```

怎样向文件读写字符

【例10.2】将一个磁盘文件中的信息复制到另一个磁盘文件中。今要求将上例建立的file1.dat文件中的内容复制到另一个磁盘文件file2.dat中。

在访问磁盘文件时，是逐个字符(字节)进行的，为了知道当前访问到第几个字节，系统用“文件读写位置标记”来表示当前所访问的位置。开始时“文件读写位置标记”指向第1个字节，每访问完一个字节后，当前读写位置就指向下一个字节，即当前读写位置自动后移。

为了知道对文件的读写是否完成，只须看文件读写位置是否移到文件的末尾。




```
#include <stdio.h>
#include <stdlib.h>
int main()
{     FILE *in,*out;                          //定义指向FILE类型文件的指针变量
      char ch,infile[10],outfile[10];         //定义两个字符数组，分别存放两个数据文件名
      printf("输入读入文件的名字:");
      scanf("%s",infile);                     //输入一个输入文件的名字
      printf("输入输出文件的名字:");
      scanf("%s",outfile);                    //输入一个输出文件的名字
      if((in=fopen(infile,"r"))==NULL)        //打开输入文件
      {     printf("无法打开此文件\n"); exit(0);   }
      if((out=fopen(outfile,"w"))==NULL)      //打开输出文件
      {     printf("无法打开此文件\n"); exit(0);   }
      ch=fgetc(in);                           //从输入文件读入一个字符，赋给变量ch
      while(!feof(in))                        //如果未遇到输入文件的结束标志
      {     fputc(ch,out);                    //将ch写到输出文件
            putchar(ch);                      //将ch显示到屏幕上
            ch=fgetc(in);                     //再从输入文件读入一个字符，赋给变量ch
      }
      putchar(10);                            //显示完全部字符后换行
      fclose(in);                             //关闭输入文件
      fclose(out);                            //关闭输出文件
      return 0;
}
```

怎样向文件读写一个字符串

读写一个字符的函数:

函数名	调用形式	功能	返回值
fgets	fgets(str,n,fp)	从fp指向的文件读入一个长度为(n-1)的字符串，存放到字符数组str中	读成功，返回地址str，失败则返回NULL
fputs	fputs(str,fp)	把str所指向的字符串写到文件指针变量fp所指向的文件中	输出成功，返回0；否则返回非0值

fgets中最后一个字母s表示字符串(string)。见名知义，fgets的含义是： 从文件读取一个字符串。

怎样向文件读写一个字符串

fgets函数的函数原型为

```
char *fgets(char*str, int n, FILE*fp);
```

其作用是从文件读入一个字符串。调用时可以写成下面的形式:

```
fgets(str,n,fp);
```

其中,n是要求得到的字符个数，但实际上只从fp所指向的文件中读入n-1个字符，然后在最后加一个'\0'字符，这样得到的字符串共有n个字符，把它们放到字符数组str中。如果在读完n－1个字符之前遇到换行符“\n”或文件结束符EOF，读入即结束，但将所遇到的换行符“\n”也作为一个字符读入。若执行fgets函数成功，则返回值为str数组首元素的地址，如果一开始就遇到文件尾或读数据出错，则返回NULL。

怎样向文件读写一个字符串

fputs函数的函数原型为

```
int fputs (char *str, FILE *fp);
```

其作用是将str所指向的字符串输出到fp所指向的文件中。调用时可以写成:

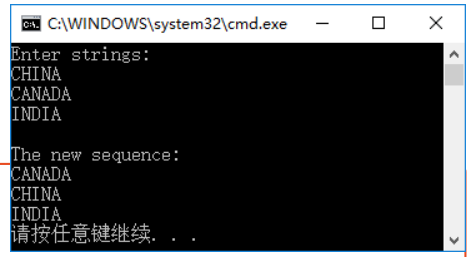
```
fputs("China",fp);
```

把字符串"China"输出到fp指向的文件中。fputs函数中第一个参数可以是字符串常量、字符数组名或字符型指针。字符串末尾的'\0'不输出。若输出成功，函数值为0;失败时，函数值为EOF(即-1)。

fgets和fgets这两个函数的功能类似于gets和puts函数，只是gets和puts以终端为读写对象，而fgets和fputs函数以指定的文件作为读写对象。

怎样向文件读写一个字符串

【例10.3】从键盘读入若干个字符串，对它们按字母大小的顺序排序，然后把排好序的字符串送到磁盘文件中保存。

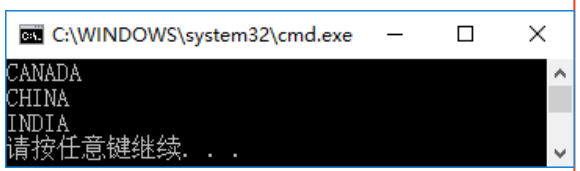



```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
int main()
{    FILE*fp;
     char str[3][10],temp[10];
     //str是用来存放字符串的二维数组，temp是临时数组
     int i,j,k,n=3;
     printf("Enter strings:\n");      //提示输入字符串
     for(i=0;i<n;i++)
          gets(str[i]);               //输入字符串
     for(i=0;i<n-1;i++)               //用选择法对字符串排序
     {    k=i;
          for(j=i+1;j<n;j++)
               if(strcmp(str[k],str[j])>0) k=j;
          if(k!=i)
          {    strcpy(temp,str[i]);
               strcpy(str[i],str[k]);
               strcpy(str[k],temp);}
     }
     if((fp=fopen("D:\\CC\\string.dat","w"))==NULL) //打开磁盘文件
     //'\'为转义字符的标志，因此在字符串中要表示'\'用'\\'
     {
          printf("can't open file!\n");
          exit(0);
     }
     printf("\nThe new sequence:\n");
     for(i=0;i<n;i++)
     {    fputs(str[i],fp);fputs("\n",fp);
          //向磁盘文件写一个字符串，然后输出一个换行符
          printf("%s\n",str[i]);       //在屏幕上显示
     }
     return 0;
}
```

【例10.3】从键盘读入若干个字符串，对它们按字母大小的顺序排序，然后把排好序的字符串送到磁盘文件中保存。

可以编写出以下的程序，从文件string.dat中读回字符串，并在屏幕上显示。

```
#include <stdio.h>
#include <stdlib.h>
int main()
{       FILE*fp;
        char str[3][10];
        int i=0;
        if((fp=fopen("D:\\CC\\string.dat","r"))==NULL)  //注意文件路径必须与前相同
        {
                printf("can't open file!\n");
                exit(0);
        }
        while(fgets(str[i],10,fp)!=NULL)
        {       printf("%s",str[i]);
                i++;
        }
        fclose(fp);
        return 0;
}
```



用格式化的方式读写文本文件

可以对文件进行格式化输入输出，这时就要用fprintf函数和fscanf函数，从函数名可以看到，它们只是在printf和scanf的前面加了一个字母f。它们的作用与printf函数和scanf函数相仿，都是格式化读写函数。只有一点不同： fprintf和fscanf函数的读写对象不是终端而是文件。它们的一般调用方式为：

fprintf(文件指针, 格式字符串, 输出表列);

fscanf(文件指针, 格式字符串, 输出表列);

```
fprintf (fp,"%d,%6.2f",i,f);      //将int型变量i和float型变量f的值按%d和%6.2f的格式输出到fp指向的文件中
```

```
fscanf (fp,"%d,%f",&i,&f);
//磁盘文件上如果有字符"3,4.5"，则从中读取整数3送给整型变量i，读取实数4.5送给float型变量f
```

用二进制方式向文件读写一组数据

C语言允许用fread函数从文件中读一个数据块，用fwrite函数向文件写一个数据块。在读写时是以二进制形式进行的。在向磁盘写数据时，直接将内存中一组数据原封不动、不加转换地复制到磁盘文件上，在读入时也是将磁盘文件中若干字节的内容一批读入内存。

```
fread(buffer, size, count, fp);
```

```
fwrite(buffer, size, count, fp);
```

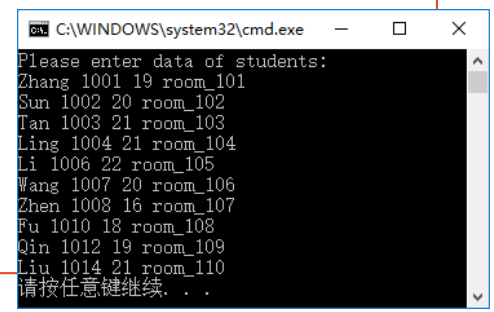
buffer:	是一个地址。对fread，它是用来存放从文件读入的数据的存储区的地址。对fwrite，是要把此地址开始的存储区中的数据向文件输出（以上指的是起始地址）。
size:	要读写的字节数。
count:	要读写多少个数据项(每个数据项长度为size)。
fp:	FILE类型指针。

```
float f[10];
fread(f,4,10,fp);     //从fp所指向的文件读入10个4个字节的数据，存储到数组f中
```

怎样向文件读写一个字符串

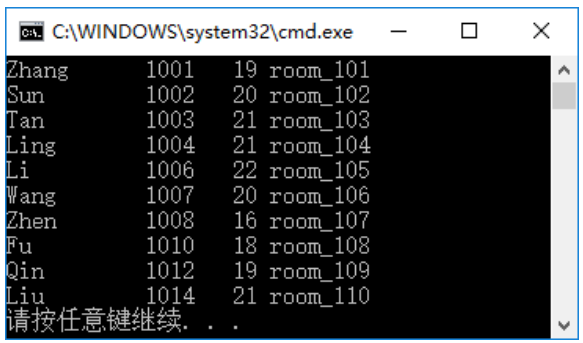
【例10.4】从键盘输入10个学生的有关数据，然后把它们转存到磁盘文件上去。

```
#include <stdio.h>
#define SIZE 10
struct Student_type
{     char name[10];
      int num;
      int age;
      char addr[15];
}stud[SIZE];    //定义全局结构体数组stud，包含10个学生数据

void save()     //定义函数save，向文件输出SIZE个学生的数据
{     FILE *fp;
      int i;
      if((fp=fopen("stu.dat","wb"))==NULL)  //打开输出文件stu.dat
      {     printf("cannot open file\n");
            return;
      }
      for(i=0;i<SIZE;i++)
            if(fwrite(&stud[i],sizeof(struct Student_type),1,fp)!=1)
                  printf("file write error\n");
      fclose(fp);
}

int main()
{     int i;
      printf("Please enter data of students:\n");
      for(i=0;i<SIZE;i++)
      //输入SIZE个学生的数据，存放在数组stud中
      scanf("%s%d%d%s",stud[i].name,&stud[i].num,
            &stud[i].age,stud[i].addr);
      save();
      return 0;
}
```



【例10.4】从键盘输入10个学生的有关数据，然后把它们转存到磁盘文件上去。

为了验证在磁盘文件stu.dat中是否已存在此数据，可以用以下程序从stu.dat文件中读入数据，然后在屏幕上输出。




```
#include <stdio.h>
#include <stdlib.h>
#define SIZE 10
struct Student_type
{      char name[10];
       int num;
       int age;
       char addr[15];
}stud[SIZE];

int main()
{      int i;
       FILE *fp;
       if((fp=fopen("stu.dat","rb"))==NULL)   //打开输入文件stu.dat
       {      printf("cannot open file\n");
              exit(0);
       }
       for(i=0;i<SIZE;i++)
       {      fread(&stud[i],sizeof(struct Student_type),1,fp);  //从fp指向的文件读入一组数据
              printf("%-10s %4d %4d %-15s\n",stud[i].name,stud[i].num,stud[i]. age,stud[i].addr);
              //在屏幕上输出这组数据
       }
       fclose(fp);                                  //关闭文件stu_list
       return 0;
}
```

【例10.4】从键盘输入10个学生的有关数据，然后把它们转存到磁盘文件上去。

从磁盘文件stu_list中读二进制数据，并存放在stud数组中。

```
##include <stdio.h>
#define SIZE 10
struct Student_type
{    char name[10];
     int num;
     int age;
     char addr[15];
}stud[SIZE];    //定义全局结构体数组stud，包含10个学生数据
void load()
{    FILE *fp;
     int i;
     if((fp=fopen("stu_list","rb"))==NULL) //打开输入文件stu_list
     {    printf("cannot open infile\n");
          return;
     }
     for(i=0;i<SIZE;i++)
          if(fread(&stud[i],sizeof(struct Student_type),1,fp)!=1)
          //从stu_ list文件中读数据
          {    if(feof(fp))
               {    fclose(fp);
                    return;
               }
                    printf("file read error\n");
               }
          fclose(fp);
}
void save()    //定义函数save，向文件输出SIZE个学生的数据
{    FILE *fp;
     int i;
     if((fp=fopen("stu.dat","wb"))==NULL)  //打开输出文件stu.dat
     {    printf("cannot open file\n");
          return;
     }
     for(i=0;i<SIZE;i++)
          if(fwrite(&stud[i],sizeof(struct Student_type),1,fp)!=1)
               printf("file write error\n");
     fclose(fp);
}
int main()
{    int i;
     load();
     save();
     return 0;
}
```

怎样向文件读写一个字符串

(1) 数据的存储方式

- 文本方式: 数据以字符方式(ASCII代码)存储到文件中。如整数12，送到文件时占2个字节，而不是4个字节。以文本方式保存的数据便于阅读。
- 二进制方式: 数据按在内存的存储状态原封不动地复制到文件。如整数12，送到文件时和在内存中一样占4个字节。

(2) 文件的分类

- 文本文件(ASCII文件): 文件中全部为ASCII字符。
- 二进制文件: 按二进制方式把在内存中的数据复制到文件的，称为二进制文件，即映像文件。

(3) 文件的打开方式

- 文本方式: 不带b的方式，读写文件时对换行符进行转换。
- 二进制方式: 带b的方式，读写文件时对换行符不进行转换。

(4) 文件读写函数

- 文本读写函数: 用来向文本文件读写字符数据的函数（如fgetc，fgets,fputc,fputs,fscanf,fprintf等）。
- 二进制读写函数: 用来向二进制文件读写二进制数据的函数（如getw,putw,fread,fwrite等）。

随机读写数据文件

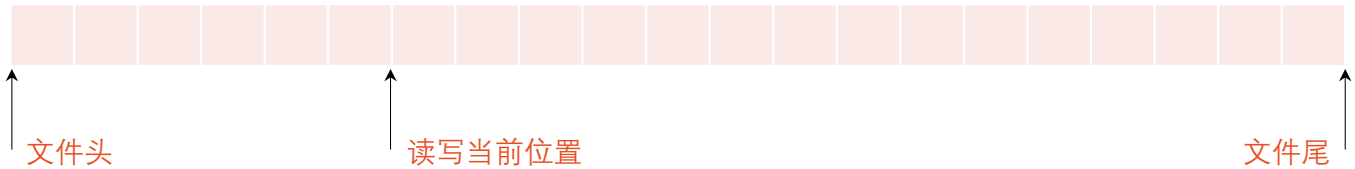
对文件进行顺序读写比较容易理解，也容易操作，但有时效率不高。而随机访问不是按数据在文件中的物理位置次序进行读写，而是可以对任何位置上的数据进行访问，显然这种方法比顺序访问效率高得多。

文件位置标记及其定位

1. 文件位置标记

为了对读写进行控制，系统为每个文件设置了一个**文件读写位置标记**(简称文件位置标记或文件标记)，用来指示“接下来要读写的下一个字符的位置”。

一般情况下，在对字符文件进行顺序读写时，文件位置标记指向文件开头，这时如果对文件进行读/写的操作，就读/写完第1个字符后，文件位置标记顺序向后移一个位置，在下一次执行读/写操作时，就将位置标记指向的第2个字符进行读出或写入。依此类推，直到遇文件尾，，此时文件位置标记在最后一个数据之后。



对流式文件既可以进行顺序读写，也可以进行随机读写。关键在于控制文件的位置标记。如果文件位置标记是按字节位置顺序移动的，就是顺序读写。如果能将文件位置标记按需要移动到任意位置，就可以实现随机读写。所谓随机读写，是指读写完上一个字符（字节）后，并不一定要读写其后续的字符（字节），而可以读写文件中任意位置上所需要的字符（字节）。即对文件读写数据的顺序和数据在文件中的物理顺序一般是不一致的。可以在任何位置写入数据，在任何位置读取数据。

文件位置标记及其定位

2. 文件位置标记的定位

(1) 用rewind函数使文件位置标记指向文件开头

```
rewind(文件指针);
```

rewind函数的作用是使文件位置标记重新返回文件的开头，此函数没有返回值。

(2) 用fseek函数改变文件位置标记

```
fseek(文件类型指针, 位移量, 起始点);
```

“起始点”：用0，1或2代替，0代表“文件开始位置”，1为“当前位置”，2为“文件末尾位置”

“位移量”：指以“起始点”为基点，向前移动的字节数（长整型）

fseek函数一般用于二进制文件。

```
fseek (fp,100L,0);          //将文件位置标记向前移到离文件开头100个字节处
fseek (fp,50L,1);           //将文件位置标记向前移到离当前位置50个字节处
fseek (fp,-10L,2);          //将文件位置标记从文件末尾处向后退10个字节
```

(3) 用ftell函数测定文件位置标记的当前位置

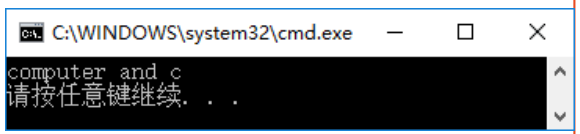
ftell函数的作用是得到流式文件中文件位置标记的当前位置，用相对于文件开头的位移量来表示。如果调用函数时出错（如不存在fp指向的文件），ftell函数返回值为-1L。

```
i=ftell(fp);                //变量i存放文件当前位置
if(i==-1L) printf("error\n");   //如果调用函数时出错，输出"error"
```

文件位置标记及其定位

【例10.5】有一个磁盘文件，内有一些信息。要求第1次将它的内容显示在屏幕上，第2次把它复制到另一文件上。

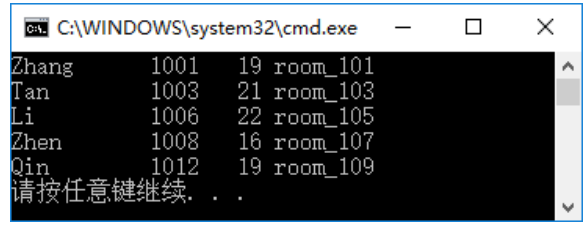
```
#include<stdio.h>
int main()
{       char ch;
        FILE *fp1,*fp2;
        fp1=fopen("file1.dat","r");     //打开输入文件
        fp2=fopen("file2.dat","w");     //打开输出文件
        ch=getc(fp1);                   //从file1.dat文件读入第一个字符
        while(!feof(fp1))               //当未读取文件尾标志
        {       putchar(ch);            //在屏幕输出一个字符
                ch=getc(fp1);           //再从file1.dat文件读入一个字符
        }
        putchar(10);                    //在屏幕执行换行
        rewind(fp1);                    //使文件位置标记返回文件开头
        ch=getc(fp1);                   //从file1.dat文件读入第一个字符
        while(!feof(fp1))               //当未读取文件尾标志
        {       fputc(ch,fp2);          //向file2.dat文件输出一个字符
                ch=fgetc(fp1);          //再从file1.dat文件读入一个字符
        }
        fclose(fp1);fclose(fp2);
        return 0;
}
```



随机读写

【例10.6】在磁盘文件上存有10个学生的数据。要求将第1,3,5,7,9个学生数据输入计算机，并在屏幕上显示出来。

```
#include<stdio.h>
#include <stdlib.h>
struct Student_type      //学生数据类型
{     char name[10];
      int num;
      int age;
      char addr[15];
}stud[10];
int main()
{     int i;
      FILE *fp;
      if((fp=fopen("stu.dat","rb"))==NULL)   //以只读方式打开二进制文件
      {     printf("can not open file\n");
            exit(0);
      }
      for(i=0;i<10;i+=2)
      {     fseek(fp,i*sizeof(struct Student_type),0);     //移动文件位置标记
            fread(&stud[i],sizeof(struct Student_type),1,fp);  //读一个数据块到结构体变量
            printf("%-10s %4d %4d %-15s\n",stud[i].name,stud[i].num,stud[i].age,stud[i].addr);
            //在屏幕输出
      }
      fclose(fp);
      return 0;
}
```



文件读写的出错检测

文件读写的出错检测

1. ferror函数 `ferror(fp);`

在调用各种输入输出函数（如putc,getc,fread,fwrite等）时，如果出现错误，除了函数返回值有所反映外，还可以用ferror函数检查。

如果ferror返回值为0（假），表示未出错;

如果返回一个非零值，表示出错。

注意

对同一个文件每一次调用输入输出函数，都会产生一个新的ferror函数值，因此，应当在调用一个输入输出函数后立即检查ferror函数的值，否则信息会丢失。

在执行fopen函数时，ferror函数的初始值自动置为0。

2. clearerr函数

clearerr的作用是使文件出错标志和文件结束标志置为0。

假设在调用一个输入输出函数时出现错误，ferror函数值为一个非零值。应该立即调用clearerr(fp)，使ferror(fp)的值变成0，以便再进行下一次的检测。

只要出现文件读写出错标志，它就一直保留，直到对同一文件调用clearerr函数或rewind函数，或任何其他一个输入输出函数。