



Chap 1 引言

1.1 一个C语言程序

1.2 程序与程序设计语言

1.3 C语言的发展历史与特点

1.4 实现问题求解的过程

本章要点

- 什么是程序？程序设计语言包含哪些功能？
- 程序设计语言在语法上包含哪些内容？
- 结构化程序设计有哪些基本的控制结构？
- C语言有哪些特点？
- C语言程序的基本框架如何？
- 形成一个可运行的C语言程序需要经过哪些步骤？
- 如何用流程图描述简单的算法？

1.1 一个C语言程序

例1-1求阶乘问题。输入一个正整数n，输出n!。

```
#include <stdio.h> /* 编译预处理命令 */
int main(void) /* 主函数 */
{
    int n; /* 变量定义 */
    int factorial(int n); /* 函数声明 */
    scanf("%d", &n); /* 输入一个整数 */
    printf("%d\n", factorial(n)); /* 调用函数计算阶乘 */
    return 0;
}
int factorial(int n) /* 定义计算 n! 的函数 */
{
    int i, fact = 1;
    for(i = 1; i <= n; i++){ /* 循环 */
        fact = fact * i;
    }
    return fact;
}
```

输入 4
输出 24

输入 13
输出

C程序由函数组成
有且只有一个主函数main()

1.2 程序与程序设计语言

■ 程序

- 人们为解决某种问题用计算机可以识别的代码编排的一系列加工步骤。
- 程序的执行过程实际上是对程序所表达的数据进行处理的过程。

■ 程序设计语言

- 提供了一种表达数据与处理数据的功能
- 要求程序员按照语言的规范编程



1.2 程序与程序设计语言

- 1.2.1 程序与指令
- 1.2.2 程序设计语言的功能
- 1.2.3 程序设计语言的语法
- 1.2.4 程序的编译与编程环境

1.2.1 程序与指令

- 指令：计算机的一个最基本的功能
如实现一次加法运算或实现一次大小的判别
- 计算机的指令系统：计算机所能实现的指令的集合
- 程序：一系列计算机指令的有序组合

程序与指令

例1-2 编写程序，分别求和与乘积

■ 虚拟的计算机指令系统（7条指令）

- 指令1: **Input X** 将当前输入数据存储在内存的X单元
- 指令2: **Output X** 将内存X单元的数据输出。
- 指令3: **Add X Y Z** 将内存X单元的数据与Y单元的数据相加并将结果存储在Z单元。
- 指令4: **Sub X Y Z** 将内存X单元的数据与Y单元的数据相减并将结果存储在Z单元。
- 指令5: **BranchEq X Y P** 比较X与Y，若相等则程序跳转到P处执行，否则继续执行下一条指令。
- 指令6: **Jump P** 程序跳转到P处执行。
- 指令7: **Set X Y** 将内存Y单元的值设为X。

程序与指令

- 输入3个数**A**, **B**和**C**, 求**A+B+C**的结果

Input A;

输入第**1**个数据到存储单元**A**中

Input B;

Input C;

Add A B D;

将**A**、**B**相加并将结果存在**D**中

Add C D D;

将**C**、**D**相加并将结果存在**D**中

Output D;

输出**D**的内容

程序与指令

- 输入A，求 $A+A+A$ 的结果

解1:

Input A;

Add A A D;

Add A D D;

Output D;

解2:

Input A;

Set 0 Z;

Add Z A Z;

Add Z A Z;

Add Z A Z;

Output Z;

程序与指令

■ 输入2个数A和B，求A*B

$A*B = A+A+.....+A$ (B个A相加)

◆ 分别输入两个数到A、B
两个变量

◆ 设X = 0, Z = 0

◆ 当X不等于B时，重复做
以下操作：

$Z = Z + A;$

$X = X + 1;$

◆ 输出Z

1. Input A;
2. Input B;
3. Set 0 X;
4. Set 0 Z;
5. BranchEq X B 9;
6. Add Z A Z;
7. Add 1 X X;
8. Jump 5;
9. Output Z;

1.2.2 程序设计语言的功能

- 数据表达：表达所要处理的数据
- 流程控制：表达数据处理的流程

数据表达

- 数据表达：一般将数据抽象为若干类型
 - 数据类型：对某些具有共同特点的数据集合的总称
 - 代表的数字（数据类型的定义域）
 - 在这些数据上做些什么（即操作或称运算）
- 例如：整数类型
- 包含的数据： $\{\dots, -2, -1, 0, 1, 2, \dots\}$
 - 作用在整数上的运算： $+ - * / \% \dots$

数据表达

■ C语言提供的数据类型

- 基本数据类型：程序设计语言事先定义好，供程序员直接使用，如整型、实型（浮点型）、字符型等。
- 构造类型：由程序员构造，如数组、结构、文件、指针等。

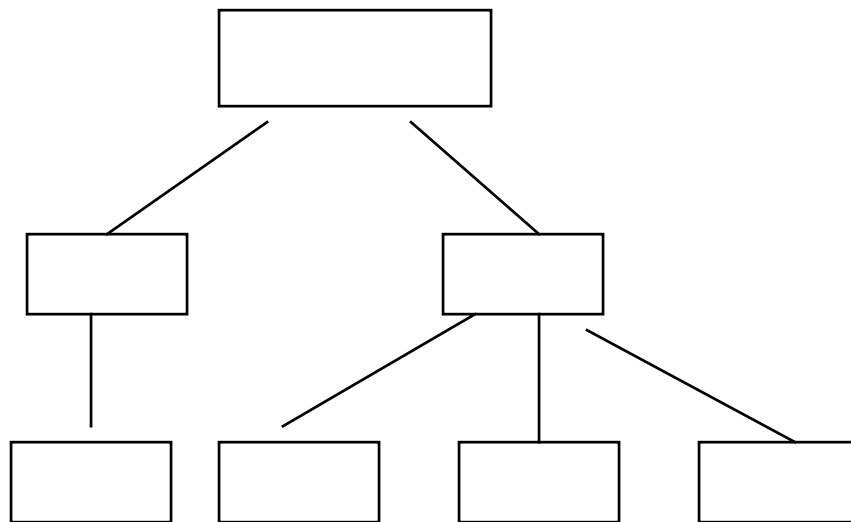
■ 各种数据类型的常量与变量形式

- 常量（常数）与变量

流程控制

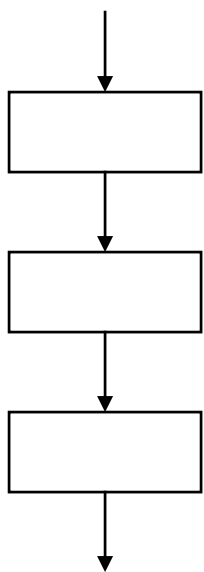
■ 结构化程序设计方法

- 将复杂程序划分为若干个相互独立的模块
- 模块：一条语句（**Statement**）、一段程序或一个函数（子程序）等
- 单入口、单出口

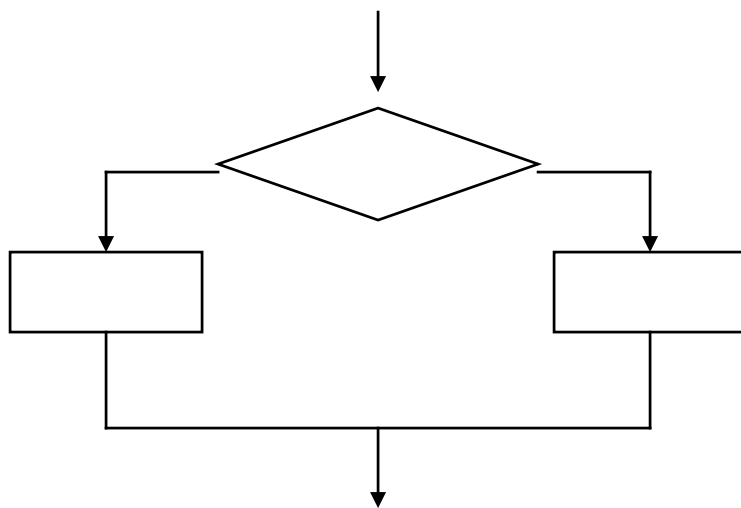


流程控制

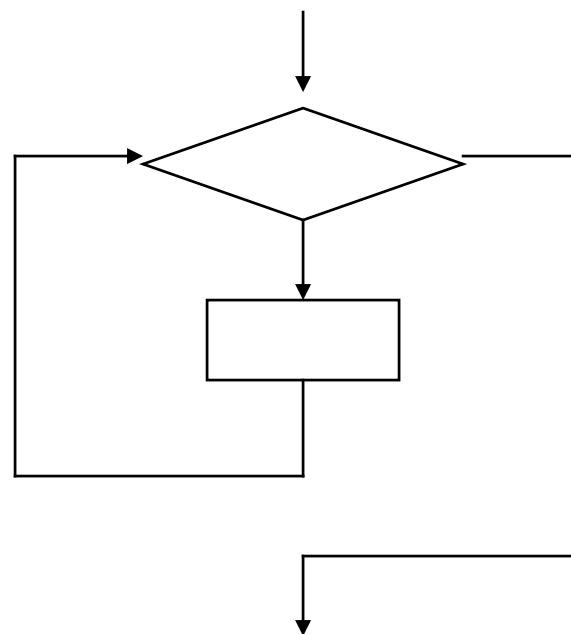
- 任何程序都可以将模块通过**3**种基本的控制结构进行组合来实现



顺序结构



分支结构



循环结构

流程控制

■ 语句级控制：3种基本的控制结构

- 顺序控制结构：自然顺序执行
- 分支控制结构（选择结构）：根据不同的条件来选择所要执行的模块
- 循环控制结构：重复执行某个模块

■ 单位级控制：函数的定义与调用

- 处理复杂问题时，将程序分为若干个相对独立的子程序（函数）

1.2.3 程序设计语言的语法

- 用程序设计语言所写的程序必须符合相应语言的语法
 - 源程序（源代码）是一个字符序列，这些字符序列按顺序组成了一系列“单词”，“单词”的组合就形成了语言有意义的语法单位，一些简单语法单位的组合又形成了更复杂的语法单位，最后一系列语法单位组合成程序。

程序设计语言的语法

■ C语言的主要“单词”

- (1) **标识符**: C语言的标识符规定由字母、数字以及下划线组成, 且第一个字符必须是字母或下划线。
- (2) **保留字(关键字)**: 它们是C语言规定的、赋予它们以特定含义、有专门用途的标识符。
- (3) **自定义标识符**: 包括在程序中定义的变量名、数据类型名、函数名以及符号常量名。**有意义的英文单词**
- (4) **常量**: 常量是有数据类型的, 如, **123**、**12.34**
- (5) **运算符**。代表对各种数据类型实际数据对象的运算。如, **+** (加)、**-** (减)、***** (乘)、**/** (除)、**%** (求余)、**>** (大于)

程序设计语言的语法

■ C语言的主要语法单位

- (1) 表达式: 运算符与运算对象组合就形成了表达式。如, **$2 + 3 * 4$**
- (2) 变量定义: 变量也有数据类型, 所以在定义变量时要说明相应变量的类型。如: **`int i;`**
- (3) 语句: 语句是程序最基本的执行单位, 程序的功能就是通过对一系列语句的执行来实现的。
- (4) 函数定义与调用

1.2.4 程序的编译与编程环境

■ 程序的编译

编译器

程序 -----> 计算机直接能理解的指令序列

编译器：对源程序进行词法分析、语法与语义分析，生成可执行的代码。

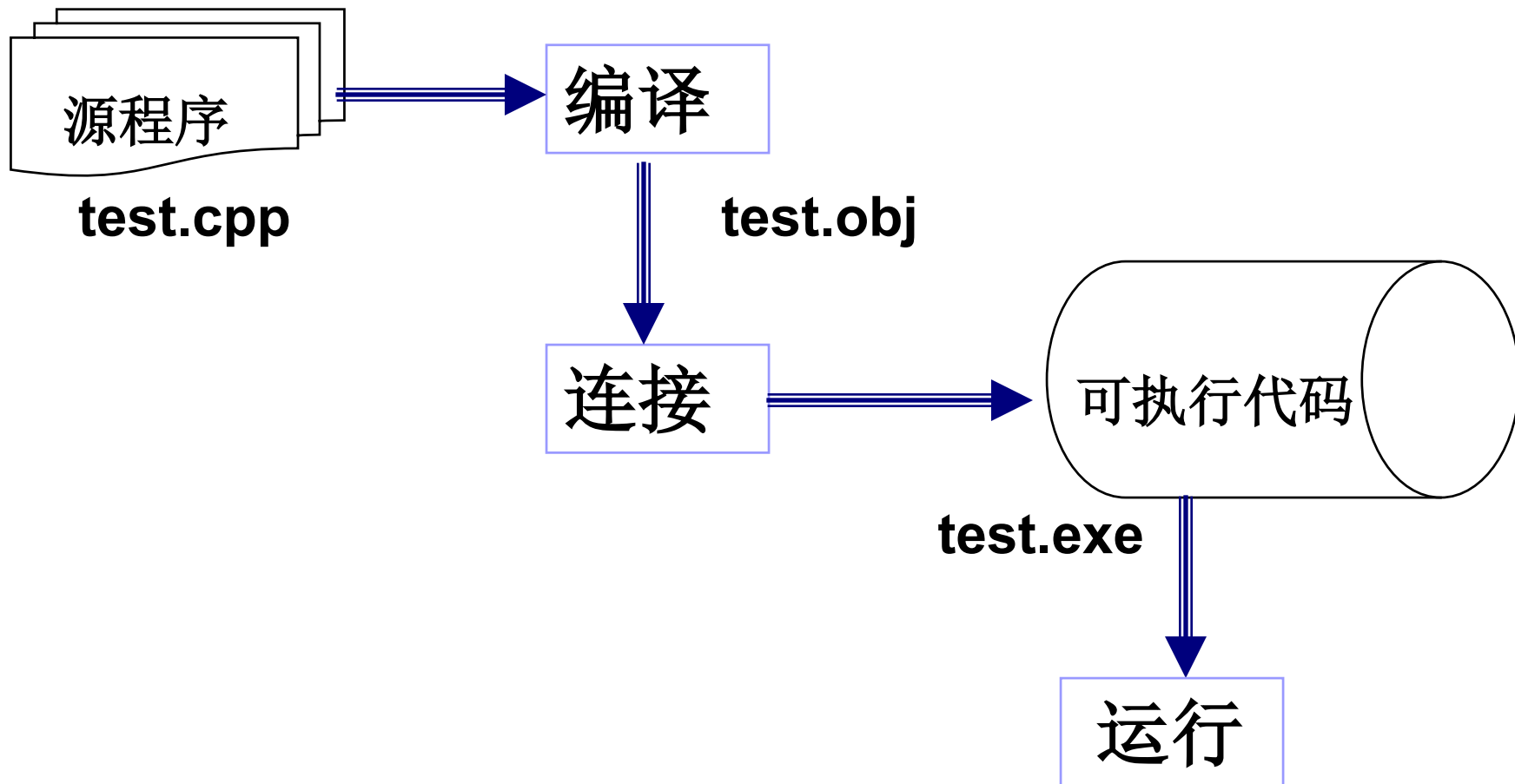
直接指出程序中的语法错误

■ 编程环境

包括编辑程序（**Edit**）、编译（**Compile**）、调试（**Debug**）等过程。

■ 掌握程序设计语言：根据语言的语法，用语言表达数据、实现程序的控制，并会使用编程环境。

C 语言上机过程



1.3 C语言的发展历史与特点

■ 历史

- **1972年：** 贝尔实验室的**Dennis Ritchie**在**B语言**的基础上设计并实现了**C语言**。
- **1978年：** **B.W.Kernighan**和**D.Ritchie**（简称**K & R**）合著的《**The C Programming Language**》是各种**C语言**版本的基础，称之为**旧标准C语言**。
- **1983年：** 美国国家标准化协会（**ANSI**）制定了新的**C语言**标准，称**ANSI C**。

C语言的特点

1. **C语言是一种结构化语言**
2. **C语言语句简洁、紧凑，使用方便、灵活**
32个关键字，9种控制语句，程序书写形式自由。
3. **C语言程序易于移植**
C语言将与硬件有关的因素从语言主体中分离出来，通过库函数或其他实用程序实现它们。
4. **C语言有强大的处理能力**
5. **生成的目标代码质量高，运行效率高**
6. **数据类型检查不严格，表达式出现二义性，不具备数据越界自动检查功能，运算符的优先级与结合性对初学者难于掌握。**

C语言中大小写字母代表不同含义

1.4 实现问题求解的过程

问题：求1~100间所有偶数的和。

1. 问题分析与算法设计

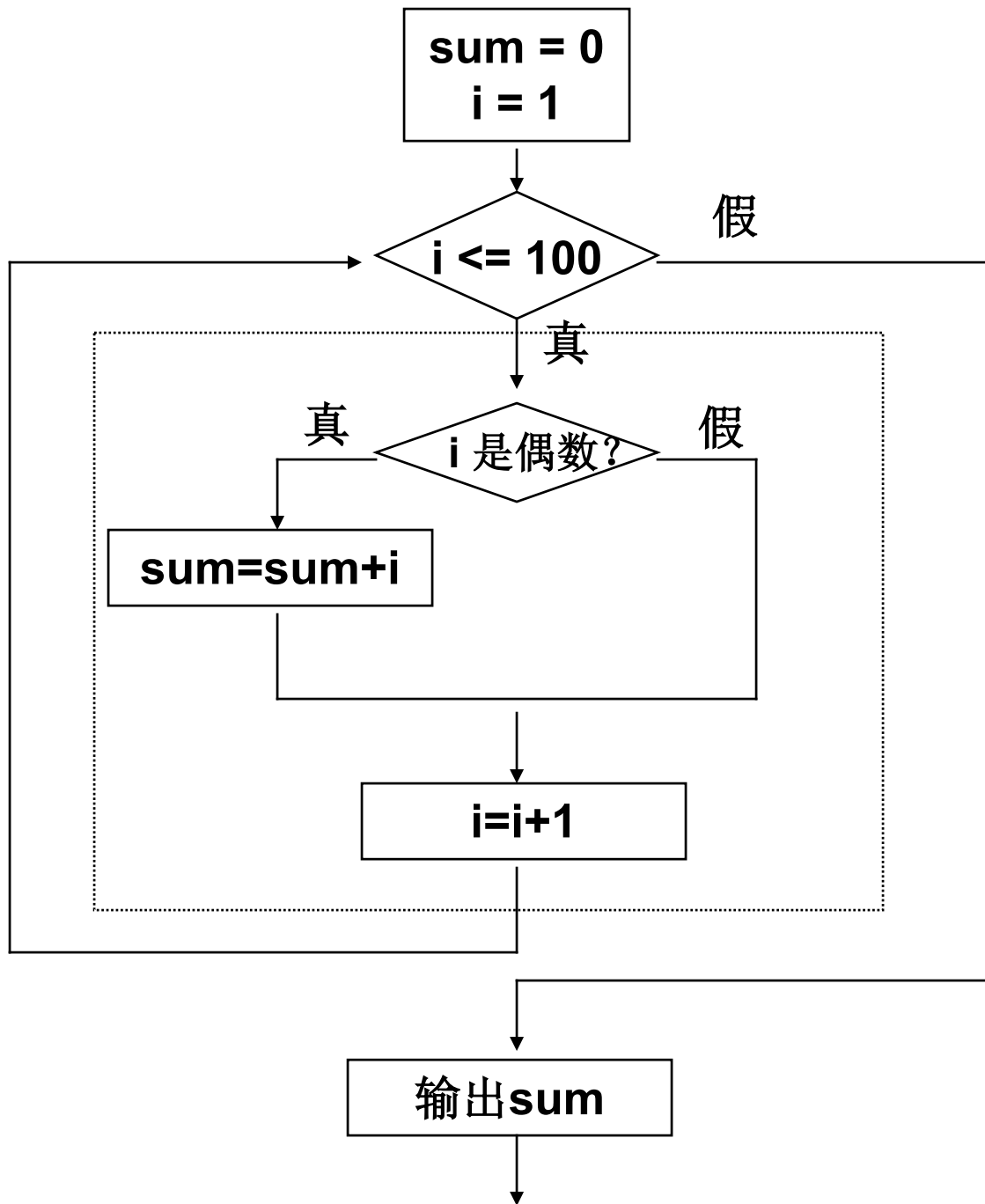
求在一定范围内（1~100）、满足一定条件(偶数)的若干整数的和，求累加和。

思路： 设置一个变量(**sum**)，其初值为0，然后在1~100的数中(**i**)寻找偶数，将它们一个一个累加到**sum**中。

- 一步累加： **sum = sum + i;**
- 重复累加，用循环语句实现，在循环过程中：
 - (1) 判别 **i** 是不是偶数：用分支控制语句来实现。
 - (2) 对循环次数进行控制：通过 **i** 值的变化

1. 问题分析与算法设计

- 思路 → 确定算法
- 算法：一组明确的解决问题的步骤，它产生结果并可在有限的时间内终止。
- 算法的描述：
 - 自然语言
 - 伪代码
 - 流程图：算法的图形表示法



2. 编辑程序

生成程序的源文件，C语言源文件的后缀为 **.c / .cpp**

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

```
int main(void)
```

```
{
```

```
    int i, sum = 0;
```

```
    for(i = 1; i <= 100; i++) {
```

```
        if (i%2 == 0) {
```

```
            sum = sum + i;
```

```
        }
```

```
    }
```

```
    printf("%d", sum);
```

```
    return 0;
```

```
}
```

3. 程序编译连接

编辑程序后，用该语言的编译程序对其进行编译，以生成二进制代码表示的目标程序(.obj)，与编程环境提供的库函数进行连接(Link)形成可执行的程序(.exe)。

编译程序指出语法错误

4. 运行与调试

经过编辑、编译、连接，生成执行文件后，就可以在编程环境或操作系统环境中运行该程序。

如果程序运行所产生的结果不是你想要的结果，这是程序的**语义错误（逻辑错误）**。

语法错误 VS 逻辑错误

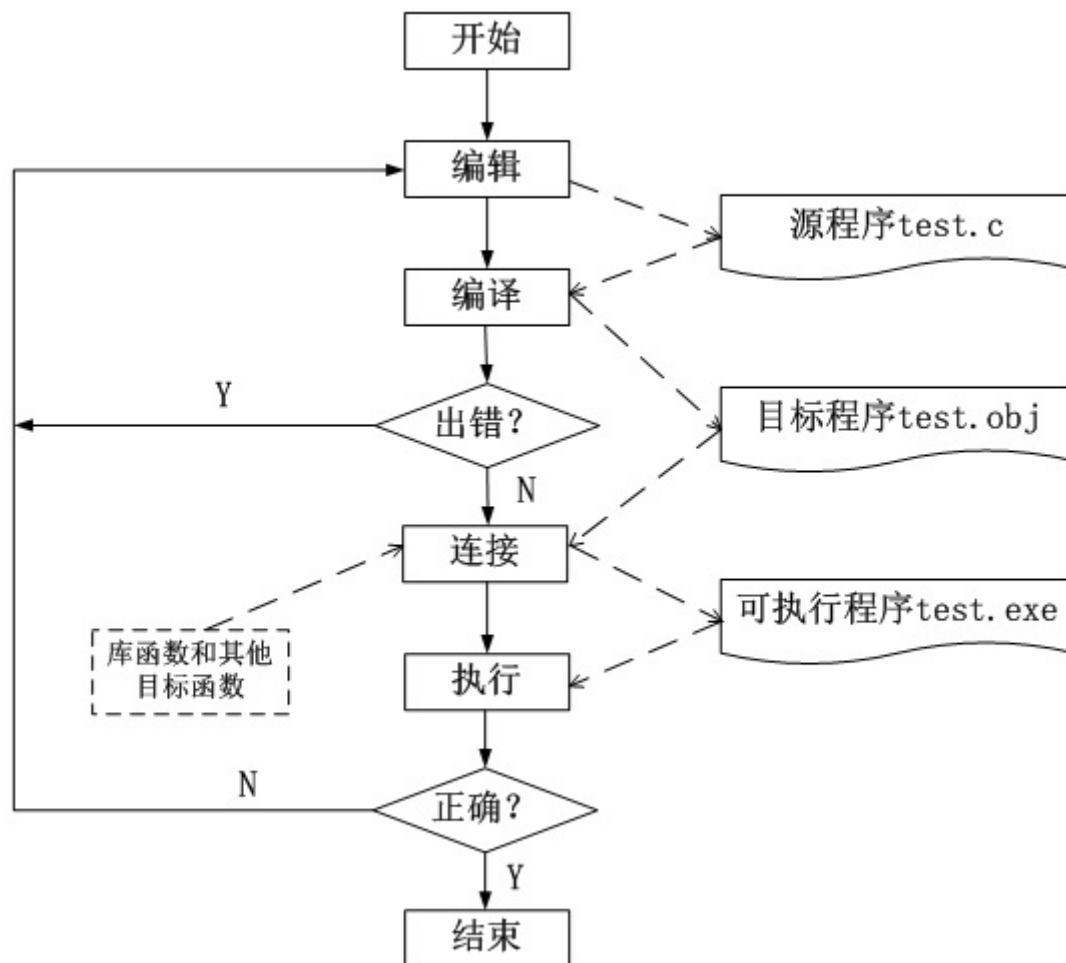
调试：在程序中查找错误并修改错误的过程。

调试的方法

- 设置断点
- 单步跟踪

调试是一个需要耐心和经验的工作，也是程序设计最基本的技能之一。

C语言程序的编辑、编译连接、运行调试步骤示意图



Chap 2 用C语言编写程序

2.1 在屏幕上显示 Hello World!

2.2 求华氏温度 100°F 对应的摄氏温度

2.3 计算分段函数

2.4 输出华氏—摄氏温度转换表

2.5 生成乘方表与阶乘表

本章要点

- 怎样编写程序，在屏幕上显示一些信息？
- 怎样编写程序，实现简单的数据处理，例如将华氏温度转换为摄氏温度？
- 怎样使用 **if** 语句计算分段函数？
- 怎样用 **for** 语句求 $1+2+\dots+100$ ？
- 如何定义和调用函数生成一张乘方表？



2.1 在屏幕上显示Hello World!

例2-1 在屏幕上显示一个短句:

Hello World!

在屏幕上显示Hello World!

```
/* 显示 “Hello World!” */  
# include <stdio.h>  
int main (void)  
{  
    printf ("Hello World! \n");  
    return 0;  
}
```

← 注释文本

← 主函数

← 语句结束

↑ 输出函数

↑ 换行符

- 1.任何程序都有主函数
- 2.程序由若干语句组成
- 3.语句由 ; 结束

在屏幕上显示一些信息

例2-2 在屏幕上显示:

Programming is fun!

And Programming in C is even more fun!

include <stdio.h> ← 编译预处理命令

int main (void)

{

printf ("Programming is fun! \n");

printf ("And Programming in C is even more fun! \n");

return 0;

}

2.2 求华氏温度 100°F 对应的摄氏温度

摄氏温度 $c = (5/9)(f-32)$

2.2.1 程序解析

2.2.2 常量、变量和数据类型

2.2.3 算术运算和赋值运算

2.2.4 格式化输出函数 `printf()`

2.2.1 程序解析

例2-3 求华氏温度 100°F 对应的摄氏温度。

摄氏温度 $c=5*(f-32)/9$

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

```
int main (void)
```

```
{
```

```
    int celsius, fahr;
```

变量定义

```
    fahr = 100;
```

```
    celsius = 5 * (fahr - 32) / 9;
```

} 变量使用

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

输出结果

```
    return 0;
```

```
}
```

输出: fahr = 100, celsius = 37

2.2.2 常量、变量和数据类型

```
int celsius, fahr;  
celsius = 5 * (fahr - 32) / 9;
```

■ 数据

- 常量：在程序运行过程中，其值不能被改变
- 变量：在程序运行过程中，其值可以被改变

■ 数据类型

- 常量：**5** 和 **9** 是整型常量（整数）
- 变量：在定义时指定

变量的定义

变量名：小写字母；见名知义

变量定义的一般形式：

类型名 变量名表；

例如：

int celsius, fahr;

定义整型变量

float x;

定义单精度浮点型变量

double area, length;

定义双精度浮点型变量

double型数据比**float**精度高，取值范围大

变量的定义

- 定义变量时要指定变量名和数据类型
类型名 变量名表;
int celsius, fahr;
float x;
double area, length;
- 变量名代表内存中的一个存储单元
存放该变量的值
- 该存储单元的大小由变量的数据类型决定
- C语言中的变量代表保存数据的存储单元
- 数学中的变量代表未知数
 $x = x + 1$

变量的定义与使用

变量必须先**定义**，后**使用**。

应该先赋值，后引用

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

```
int main (void)
```

```
{
```

```
    int celsius, fahr;
```

```
    fahr = 100;
```

```
    celsius = 5 * (fahr - 32) / 9;
```

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

```
    return 0;
```

```
}
```

一个变量名只能定义一次
变量一般都定义在程序的头上
不能定义在程序的中间或后面

2.2.3 算术运算和赋值运算

```
fahr = 100;
```

```
celsius = 5 * (fahr - 32) / 9;
```

1. 算术运算

- 双目算术运算符：+ - * / %

- 算术表达式：用算术运算符将运算对象连接起来的符合C语言语法规则的式子

数学式：5(f-32) / 9

C表达式：5 * (fahr - 32) / 9 或者：

数学式：s(s-a)(s-b)(s-c)

C表达式：

算术运算

- 双目算术运算符：+ - * / %
- 算术表达式：用算术运算符将运算对象连接起来的符合C语言语法规则的式子

注意

- / 整数除整数，得整数

$$1/2 = 0, \quad 9/4 = 2$$

5 *(fahr - 32) / 9 和 5 / 9 * (fahr - 32) 等价吗？

- % 针对整型数据

$$5\%6=5, \quad 9\%4=1, \quad 100\%4=0$$

- 双目运算符两侧操作数的类型要相同

赋值运算

- 赋值运算符 **=**
- 赋值表达式：用 **=** 将一个**变量**和一个**表达式**连接起来的式子

变量 = 表达式

=的左边必须是一个变量

例如：

fahr = 100;

celsius = 5 * (fahr - 32) / 9;

- 计算赋值运算符右侧**表达式**的值
- 将赋值运算符右侧**表达式**的值赋给左侧的**变量**

2.2.4 格式化输出函数printf()

数据输出：格式化输出函数 **printf()**

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

```
printf ("Hello World! \n");
```

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

printf (格式控制字符串, 输出参数1, ..., 输出参数n);



要输出的数据

用双引号括起来，表示输出的格式

printf—格式控制字符串

```
printf ("Hello World! \n");
```

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

```
printf ("Hi\n", fahr);
```

格式控制字符串：

- 普通字符：原样输出
- 格式控制说明：按指定的格式输出数据，**%...**
与数据类型有关
 - int型：**%d**
 - float double型：**%f**

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

输出：fahr = **100**, celsius = **37**



2.3 计算分段函数

2.3.1 程序解析

2.3.2 关系运算

2.3.3 if-else语句

2.3.4 格式化输入函数scanf

2.3.5 常用数学库函数

2.3.1 程序解析

例2-4 分段计算水费

输入用户的月用水量 x （吨），计算并输出该用户应支付的水费 y （元）（保留2位小数）

$$y = f(x) = \begin{cases} \frac{4x}{3} & x \leq 15 \\ 2.5x - 10.5 & x > 15 \end{cases}$$

要解决的问题：

- 输入
- 计算分段函数
- 输出，并保留2位小数

2.3.1 程序解析—求分段函数

```
# include <stdio.h>
int main (void)
```

```
{
```

```
    double x, y;
```

```
    printf ("Enter x (x>=0):\n"); /* 输入提示 */
```

```
    scanf ("%lf", &x);
```

/* 调用scanf函数输入数据 */

```
    if ( x <= 15 ){
```

/* if – else语句 */

```
        y = 4 * x / 3;
```

数据必须输入吗？

```
    }
```

```
    else {
```

```
        y = 2.5 * x - 10.5;
```

```
    }
```

```
    printf ("f(%f) = %.2f\n", x, y);
```

```
    return 0;
```

```
}
```

Enter x (x>=0):

9.5

f(9.500000)=12.67

Enter x (x>=0):

15

f(15.000000)=20.00

Enter x (x>=0):

21.3

f(21.300000)=42.75

2.3.2 关系运算

$x \leq 15$

比较 x 和 15 的大小

比较的结果：真 假

当 x 取值9.5时， $x \leq 15$ 的结果是：？

当 x 取值21.3时， $x \leq 15$ 的结果是：？

关系运算 — 比较运算，比较两个操作数

■ 关系运算符

$x < y$ $x \leq y$ $x == y$

$x > y$ $x \geq y$ $x != y$

区分= 和==

■ 关系表达式：用关系运算符将2个表达式连接起来的式子。

如： $x \leq 1$

运用关系表达式

表示比较的数学式

$x \leq 10$

$x \geq 10$

$x \neq 10$

$x = 10$

C关系表达式

$x \leq 10$

$x \geq 10$

$x \neq 10$

$x == 10$

用关系表达式描述条件

□ 判断 x 是否为负数

$x < 0$

□ 判断 x 是否不为零

$x \neq 0$

2.3.3 if - else语句

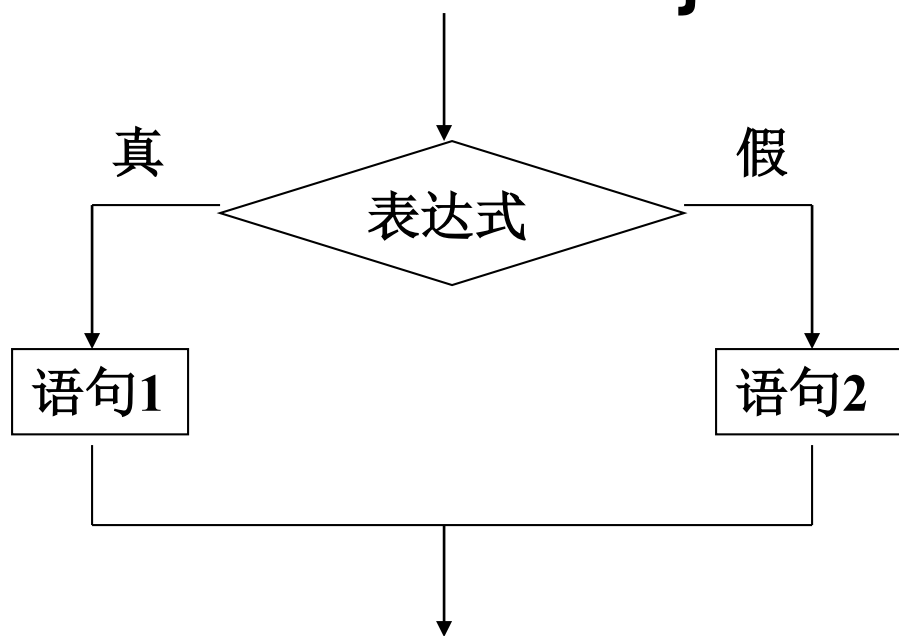
if (表达式)

语句1

else

语句2

```
if ( x <= 15 ) {  
    y = 4 * x / 3;  
}  
else {  
    y = 2.5 * x - 10.5;  
}
```



计算二分段函数

$$f(x) = \begin{cases} \frac{1}{x} & x \neq 0 \\ 0 & x = 0 \end{cases}$$

if (表达式)
语句1

else
语句2

```
if ( x != 0 ) {  
    y = 1/x;  
}  
else {  
    y = 0;  
}
```

源程序

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

```
int main (void)
```

```
{
```

```
    double x, y;
```

```
    printf ("Enter x:\n");
```

```
    scanf ("%lf", &x);
```

```
    if ( x != 0 ){
```

```
        y = 1/x;
```

```
    }
```

```
    else{
```

```
        y = 0;
```

```
    }
```

```
    printf ("f(%.2f) = %.1f\n", x, y);
```

```
    return 0;
```

```
}
```

```
# include <stdio.h> /* 例2-4 */
```

```
int main (void)
```

```
{
```

```
    double x, y;
```

```
    printf ("Enter x (x>=0):\n");
```

```
    scanf ("%lf", &x);
```

```
    if ( x <= 15 ){
```

```
        y = 4 * x / 3;
```

```
    }
```

```
    else {
```

```
        y = 2.5 * x - 10.5;
```

```
    }
```

```
    printf ("f(%f) = %.2f\n", x, y);
```

```
    return 0;
```

```
}
```

```
# include <stdio.h>
int main (void)
{
    double x, y;

    printf ("Enter x:\n");
    scanf ("%lf", &x);
    if ( x != 0 ) {
        y = 1/x;
    }
    else {
        y = 0;
    }
    printf ("f(%.2f) = %.1f\n", x, y);
    return 0;
}
```

运行结果

input x:

2.5

f(2.50)=0.4

input x:

0

f(0.00)=0.0

软件测试的基本思想

```
if ( x != 0 )  
    y = 1/x;  
else  
    y = 0;
```

input x:

2.5

f(2.50)=0.4

input x:

0

f(0.00)=0.0

软件测试

精心设计一批**测试用例** [输入数据，预期输出结果]，然后分别用这些测试用例运行程序，看程序的实际运行结果与预期输出结果是否一致。

```
if ( x <= 15 )  
    y = 4 * x / 3;  
else  
    y = 2.5 * x - 10.5;
```

Enter x (x>=0):

9.5

f(9.500000)=12.67

Enter x (x>=0):

15

f(15.000000)=20.00

Enter x (x>=0):

21.3

f(21.300000)=42.75

2.3.4 格式化输入函数scanf()

数据输入：格式化输入函数 **scanf()**

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

```
scanf ("%lf", &x);
```

scanf (格式控制字符串, 输入参数1, ..., 输入参数n);



变量地址

用双引号括起来，表示输入的格式

scanf—格式控制字符串

格式控制字符串：

- 格式控制说明：按指定的格式输入数据，**%...**
与数据类型有关

- **int**型：**%d**

- **float**型：**%f**

- **double**型：**%lf**

- 普通字符：原样输入

尽量不要出现普通字符

例如：

```
scanf ("%lf", &x);
```

```
scanf ("x=%lf", &x);
```

输入：**9.5**

输入：**x=9.5**

改进例2-3的程序

例2-3 求华氏温度 100°F 对应的摄氏温度。

摄氏温度 $C=5*(F-32)/9$

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

```
int main (void)
```

```
{
```

```
    int celsius, fahr;
```

```
    printf ("Enter fahr: \n");
```

```
    fahr = 100;
```

```
    scanf ("%d", &fahr);
```

```
    celsius = 5 * (fahr - 32) / 9;
```

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

```
    return 0;
```

```
}
```

Enter fahr:

100

fahr =100, celsius = 37

2.3.5 常用数学库函数

■ 库函数

- C语言处理系统提供事先编好的函数，供用户在编程时调用。**scanf (), printf (), exp ()**
- 在相应的系统文件（头文件）中定义一些必需的信息。

■ # include命令

- 用户调用库函数时，将相应的头文件包含到源程序中。

例如

- 调用**scanf, printf**，需要 **# include <stdio.h>**
- 调用**sqrt**，需要 **# include <math.h>**

常用数学库函数

- 平方根函数 **sqrt (x)**
- 绝对值函数 **fabs (x)**
fabs(-3.56) 的值为3.56
- 幂函数 **pow (x, n)** : x^n
pow(1.1, 2) 的值为1.21 (即1.1²)
- 指数函数 **exp (x)**: e^x
exp(2.3) 的值为 $e^{2.3}$
- 以e为底的对数函数 **log (x)**: $\ln x$
log(123.45) 的值为4.815836
- 以10为底的对数函数 **log10 (x)**: $\log_{10}x$
log10(123.45) 的值为2.091491。

例2-5 计算存款的本息

输入存款金额 **money**、存期 **year** 和年利率 **rate**，根据公式计算存款到期时的本息合计 **sum**（税前），输出时保留**2**位小数。

$$\text{sum} = \text{money} (1 + \text{rate})^{\text{year}}$$

$$\text{sum} = \text{money} * \text{pow}((1 + \text{rate}), \text{year})$$

例2-5 程序

```
# include <stdio.h>
# include <math.h>
int main (void)
{   int money, year;
    double rate, sum;

    printf ("Enter money:");
    scanf ("%d", &money);
    printf ("Enter year: ");
    scanf ("%d", &year);
    printf ("Enter rate:");
    scanf ("%lf", &rate);
    sum = money * pow ((1 + rate), year);
    printf ("sum = %.2f", sum);
    return 0;
}
```

Enter money: 1000

Enter year: 3

Enter rate: 0.025

sum = 1076.89

```
scanf ("%d%d%lf", &money, &year, &rate);
```

调用scanf函数输入多个数据

```
scanf ("%d%d%lf", &money, &year, &rate);
```

输入: 1000 3 0.025

输入参数、格式控制说明、输入数据

- **scanf**需要多个输入参数和多个格式控制说明
输入参数的类型、个数和位置要与格式控制说明一一对应

```
? scanf ("%d%lf%d ", &money, &year, &rate);
```
- 程序运行时，输入的多个数据之间必须有间隔。

```
scanf ("%d%lf%d ", &money, &rate , &year);
```

如何输入？



2.4 输出华氏—摄氏温度转换表

2.4.1 程序解析

2.4.2 for语句

2.4.3 指定次数的循环程序设计

2.4.1 程序解析

例2-6输入2个整数**lower**和 **upper**，输出一张华氏—摄氏温度转换表，华氏温度的取值范围是**[lower, upper]**，每次增加1° F。

fahr	celsius
30	-1.1
31	-0.6
32	0.0
33	0.6
34	1.1
35	1.7

2.4.1 程序解析-温度转换表

```
#include <stdio.h>
int main (void)
{ int fahr, lower, upper;
  double celsius;
  printf ("Enter lower:"); scanf ("%d", &lower);
  printf ("Enter upper:"); scanf ("%d", &upper);
  if(lower <= upper){
    printf("fahr celsius\n");
    //温度重复转换：华氏温度从lower开始，到upper结束，每次增加1° F
    for (fahr = lower ; fahr<= upper; fahr++){
      celsius = (5.0 / 9.0) * (fahr - 32);
      printf ("%4d%6.1f\n", fahr, celsius);
    }
  }else
    printf ("Invalid Value!\n");
  return 0;
}
```

Enter lower: 30

Enter upper: 35

fahr	celsius
30	-1.1
31	-0.6
32	0.0
33	0.6
34	1.1
35	1.7

fahr = fahr+1

例2-6中for语句的流程

```
for (fahr = lower; fahr <= upper; fahr++) {  
    celsius = (5.0 / 9.0) * (fahr - 32)  
    printf ("%d %6.1f\n", fahr, celsius);  
}
```

fahr = fahr+2
输出?

Enter lower: 30

Enter upper: 35

fahr	celsius
------	---------

30	-1.1
----	------

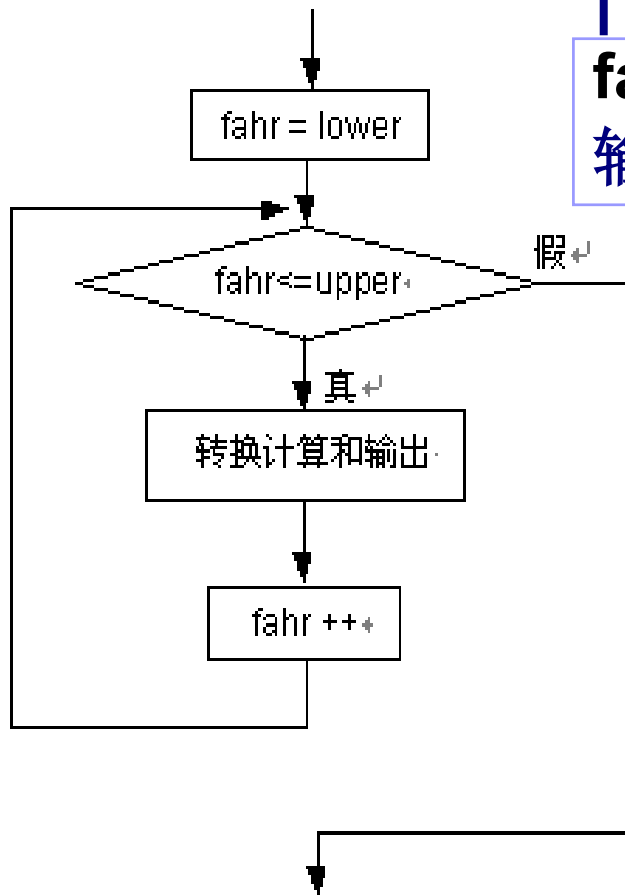
31	-0.6
----	------

32	0.0
----	-----

33	0.6
----	-----

34	1.1
----	-----

35	1.7
----	-----



2.4.2 for语句—循环语句

for (表达式1; 表达式2; 表达式3)

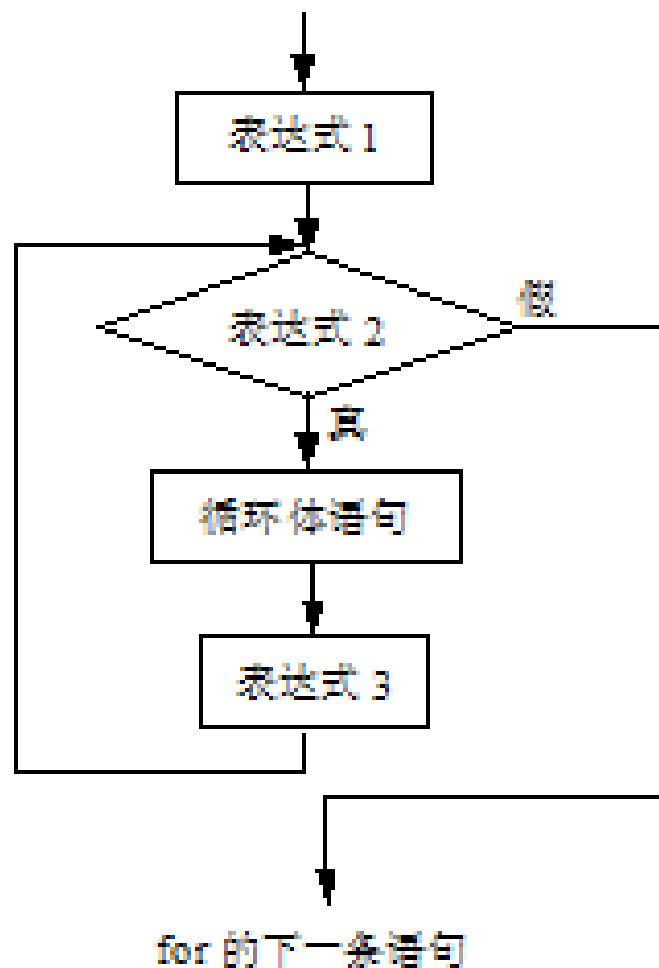
循环体语句

实现C语句的重复执行

3个表达式、循环体语句

! 书写顺序和执行顺序不同

! 表达式1只执行一次



for语句中的循环变量

循环(控制)变量：**for**语句中，通过改变或判断某个变量的值来控制循环的执行

```
for (fahr = lower; fahr <= upper; fahr ++) {  
    celsius = (5.0 / 9.0) * (fahr - 32.0);  
    printf ("%d %6.1f\n", fahr, celsius);  
}
```

↑
赋初值

↑
判断其值

↑
改变其值

for语句的说明

```
for (fahr = lower; fahr <= upper; fahr++) {  
    celsius = (5.0 / 9.0) * (fahr - 32.0);  
    printf ("%d %6.1f\n", fahr, celsius);  
}
```

表达式1： 给循环变量赋初值，指定循环的起点。

`fahr = lower`

表达式2： 给出循环的条件，判断循环是否达到终点？

`fahr <= upper`

表达式3： 设置循环的步长，改变循环变量的值，从而可改变表达式2的真假性。

`fahr++`

循环体语句： 被反复执行的语句，一条语句。

复合语句{ } and 空语句 ;

```
for (fahr = lower; fahr <= upper; fahr++) {  
    celsius = (5.0 / 9.0) * (fahr - 32.0);  
    printf ("%d %6.1f\n", fahr, celsius);  
}
```

```
for (fahr = lower ; fahr <= upper; fahr = fahr + 1)  
    celsius = (5.0 / 9.0) * (fahr - 32.0);  
    printf ("%d %6.1f\n", fahr, celsius);
```

```
for (fahr = lower ; fahr <= upper; fahr = fahr + 1);  
    celsius = (5.0 / 9.0) * (fahr - 32.0);  
    printf ("%d %6.1f\n", fahr, celsius);
```

! 不要在for语句中随意加分号

2.4.3 指定次数的循环程序设计

求 $1+2+\dots+100$

抽取具有共性的算式: $\text{sum} = \text{sum} + i$

sum 初值为0, 该算式重复100次, i 从1变到100

设 i 为循环变量, 则:

指定循环起点的表达式1: $i = 1$

给出循环条件的表达式2: $i \leq 100$

设置循环步长的表达式3: $i++$

循环体语句: $\text{sum} = \text{sum} + i;$

for ($i=1; i \leq 100; i++$)

$\text{sum} = \text{sum} + i;$

源程序 求 $1+2+\dots+100$

sum=5050

```
/* 计算  $1 + 2 + 3 + \dots + 100$  */
```

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

```
int main (void)
```

```
{
```

```
    int i, sum;
```

```
    sum = 0;
```

```
    for ( i = 1; i <= 100; i++ ){
```

```
        sum = sum + i;
```

```
    }
```

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

```
    return 0;
```

```
}
```

```
for (i=1; i<=100; i++){
```

```
    sum = 0;
```

```
    sum = sum+i;
```

```
}
```

/* 置累加和sum的初值为0 */

/* 循环重复100次 */

/* 反复累加 */

/* 输出累加和 */

求 $1+1/2+1/3+\dots+1/100$

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

```
int main(void)
```

```
{
```

```
    int i, sum;
```

```
    sum = 0;
```

```
    for ( i = 1; i <= 100; i++ ){
```

```
        sum = sum + i;
```

```
    }
```

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

```
    return 0;
```

```
}
```

```
int i;
```

```
double sum;
```

```
for (i = 1; i <= 100; i++){
```

```
    sum = sum + 1.0/i;
```

```
}
```

```
printf ("sum = %f\n", sum);
```

指定次数的循环程序设计

一般包含四个部分：

- 初始化：指定循环起点

- 给循环变量赋初值，如 $i = 1$;
- 进入循环之前，设置相关变量的初值，如 $sum = 0$ 。

- 条件控制：

- 只要 $i \leq 100$ ，循环就继续

- 工作：指重复执行的语句（循环体）。

- 一条语句，可以是复合语句或空语句。如 $sum = sum + i$ 。

- 改变循环变量：在每次循环中改变循环变量的值

- 如 $i++$ ，以改变循环条件的真假。一旦 $i > 100$ ，循环结束。

例2-7 求 $1+2+3+\dots+n$

输入一个正整数n，求前n项和，即循环n次

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

```
int main(void)
```

```
{
```

```
    int i, sum;
```

```
    sum = 0;
```

```
    for ( i = 1; i <= 100; i++ ) {
```

```
        sum = sum + i;
```

```
    }
```

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

```
    return 0;
```

```
}
```

```
printf ("Enter n:");  
scanf ("%d", &n);
```

```
for (i = 1; i<=n; i++) {  
    sum = sum + i;  
}
```

```
Enter n: 100  
Sum = 5050
```

求 $1+1/2+1/3+.....+1/n$

```
# include <stdio.h>
int main (void)
{
    int i, sum;

    printf ("Enter n: ");
    scanf ("%d", &n);
    sum = 0;
    for ( i = 1; i <= n; i++ ){
        sum = sum + i;
    }
    printf ("sum = %d\n", sum);
    return 0;
}
```

```
# include <stdio.h>
int main (void)
{ int i;
  double sum;

  printf ("Enter n: ");
  scanf ("%d", &n);
  sum = 0;
  for ( i = 1; i <= n; i++ ){
      sum = sum + 1.0/i;
  }
  printf ("sum = %f\n", sum);
  return 0;
}
```

求 $1 + 1/3 + 1/5 + \dots$ 的前n项和

求前n项和，即循环n次，每次累加1项。

```
for (i = 1; i <= n ; i++){  
    sum = sum + item (第i项)  
}
```

$item = 1.0 / (2 * i - 1)$

源程序 求 $1+1/3+1/5+\dots$

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

```
int main (void)
```

```
{  int  i, n;
```

```
    double item, sum;
```

```
    printf ("Enter n: ");
```

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

```
    sum = 0 ;
```

```
    for ( i = 1; i <= n; i++ ) {
```

```
        item = 1.0 / (2 * i - 1);
```

```
        sum = sum + item ;
```

```
    }
```

```
    printf ("sum = %f\n", sum);
```

```
    return 0;
```

```
}
```

/* 计算第i项的值 */

/* 累加第i项的值 */

例2-8 求 $1 - 1/3 + 1/5 - \dots$ 的前n项和

求前n项和，即循环n次，每次累加1项。

```
for (i = 1; i <= n ; i++){  
    sum = sum + item (第i项)  
}
```

$item = flag * 1.0 / (2 * i - 1)$

$item = flag * 1.0 / denominator$

$denominator = denominator + 2$

$flag = -flag$

$item = flag * 1.0 / denominator$

$sum = sum + item$

例2-8 源程序

```
# include <stdio.h>
int main(void)
{   int denominator, flag, i, n;
    double item, sum;
    printf ("Enter n: ");
    scanf ("%d", &n);
    flag = 1;
    denominator = 1;
    item = 1;
    sum = 0 ;
    for ( i = 1; i <= n; i++ ) {
        sum = sum + item ;
        flag = -flag;
        denominator = denominator +2;
        item = flag * 1.0/ denominator;
    }
    printf ( "sum = %f\n", sum);
    return 0;
}
```

```
/* 累加第i项的值 */
/* 准备下一次循环 */
/* 计算第i+1项的值 */
```

例2-9 求n!

$$n! = 1 * 2 * \dots * n$$

product = ?

```
for (i = 1; i <= n ; i++){  
    product = product * item (第i项)  
}
```

item = i

例2-9 源程序

```
# include <stdio.h>
int main (void)
{
    int i, n;
    double product;

    printf ("input n: \n");
    scanf ("%d", &n) ;
    product=1; /* 置阶乘product的初值为1 */
    for ( i = 1; i <= n; i++ ){ /* 循环重复n次, 计算n! */
        product = product * i ;
    }
    printf ( "product = %.0f\n", product );
    return 0;
}
```

求 x^n

输入实数 x 和正整数 n ,
计算 $x^n = x * x * \dots * x$

```
for (i = 1; i <= n ; i++){  
    power = power * item (第i项)  
}
```

item=?

源程序 求 x^n

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

```
int main(void)
```

```
{
```

```
    int i, n;
```

```
    double x, power ;
```

```
    printf ("Enter x, n: \n");
```

```
    scanf ("%lf%d", &x, &n) ;
```

```
    power = 1; /* 置power的初值为1 */
```

```
    for ( i = 1; i <= n; i++ ){ /* 循环重复n次, 计算x的n次幂 */
```

```
        power = power * x;
```

```
    }
```

```
    printf ( "%0.f\n", power);
```

```
}
```

2.5 生成乘方表和阶乘表

例2-10 生成乘方表

输入一个正整数 n ，生成一张2的乘方表，输出 2^0 到 2^n 的值，可以调用幂函数计算2的乘方。

```
for (i = 0; i <= n ; i++){  
    power = pow (2, i); /*调用幂函数pow(2,i)计算2的i次方*/  
    输出power的值;  
}
```

源程序：生成乘方表

```
# include <stdio.h>
# include <math.h>
int main (void)
{
    int i, n;
    double power;
    printf ("Enter n:");
    scanf ("%d", &n);
    for (i = 0; i <= n ; i++){
        power = pow(2, i); /* 调用幂函数pow(2,i)计算2的i次方 */
        printf ("pow(2,%d)= %.0f\n", i, power);
    }
    return 0;
}
```

Enter n: 4

pow(2,0)= 1

pow(2,1)= 2

pow(2,2)= 4

pow(2,3)= 8

pow(2,4)= 16

例2-11 生成阶乘表

输入一个正整数 n ，生成一张阶乘表，输出 $0!$ 到 $n!$ 的值。要求定义和调用函数 $\text{fact}(n)$ 计算 $n!$ ，函数类型是 double 。

```
for (i = 0; i <= n ; i++){  
    product = fact (i);    /* 调用自定义函数fact(i)计算i! */  
    输出product的值;  
}
```

```
for (i = 0; i <= n ; i++){  
    power = pow (2, i);    /*调用幂函数pow(2,i)计算2的i次方*/  
    输出power的值;  
}
```

源程序：生成阶乘表

Enter n: 3

0!=1

1!=1

2!=2

3!=6

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

```
double fact (int n); /* 自定义函数的声明 */
```

```
int main (void)
```

```
{ int i, n;  
  double result;
```

```
printf ("Enter n:");
```

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

```
for (i = 0; i <= n ; i++){
```

```
    result = fact (i); /* 调用自定义函数fact(i)计算i! */
```

```
    printf ("%d!=%.0f\n", i, result);
```

```
}
```

```
return 0;
```

```
}
```

```
double fact (int n) /* 函数首部 */
```

```
{ int i; double product;
```

```
  product = 1;
```

```
  for (i = 1; i <= n; i++) {
```

```
    product = product * i;
```

```
  }
```

```
  return product; /* 将结果回送主函数 */
```

函数的概念

- C语言中有两种类型函数

- 标准库函数
- 自定义函数

- 函数可以做到一次定义、多次调用

- 使用自定义函数的程序框架

```
double fact (int n);          /* 声明自定义函数，以分号结束 */
int main (void)
{
    .....
    result = fact (i);        /* 调用自定义函数fact(i)计算i! */
    .....
}
/* 定义求 n! 的函数 */
```

Chap 3 分支结构

本章要点

- 分支结构
 - **if-else**语句
 - **else if**/嵌套的**if-else/switch**
- **switch**语句
 - **case**后为常量表达式
 - **break**的使用
- 数据类型：**char**型
- 运算符与表达式
 - 逻辑运算符、关系运算符
 - 逻辑表达式



本章主要几个例子

3.1 简单的猜数游戏

3.2 四则运算

3.3 查询自动售货机中商品的价格

3.1 简单的猜数游戏

输入你所猜的整数（假定1~100内），与计算机产生的被猜数比较，若相等，显示猜中；若不等，显示与被猜数的大小关系

3.1.1 程序解析

3.1.2 二分支结构和if – else语句

3.1.3 多分支结构和else – if 语句

3.1.1 程序解析

例3-1 简单的猜数游戏。输入你所猜的整数 **yournumber**(假定1~100内)，与计算机产生的被猜数**mynumber**比较，若相等，显示猜中；若不等，显示与被猜数的大小关系。

yournumber vs mynubmer: 3种情况:

□ **yournumber == mynumber**

□ **yournumber > mynumber**

□ **yournumber < mynumber**

if (==) ok

else

if (>) bigger

else smaller

源程序-猜数游戏

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

```
int main (void)
```

```
{
```

```
    int mynumber = 38
```

```
    int yournumber;
```

```
    printf ("Input your number: ");
```

```
    scanf ("%d", &yournumber);
```

```
    if (yournumber == mynumber){  
        printf ("Ok! you are right!\n");}
```

```
    else{
```

```
        if (yournumber > mynumber ){
```

```
            printf ("Sorry! your number is bigger than my number!\n");}
```

```
        else{
```

```
            printf ("Sorry! your number is smaller than my number!\n");}
```

```
        }
```

```
    return 0;
```

```
}
```

Input your number:48

Sorry! your number is bigger than my number!

Input your number:38

Ok! you are right!

多层缩进的书写格式
使程序层次分明

3.1.2 二分支结构和 if-else 语句

if (表达式)

语句1

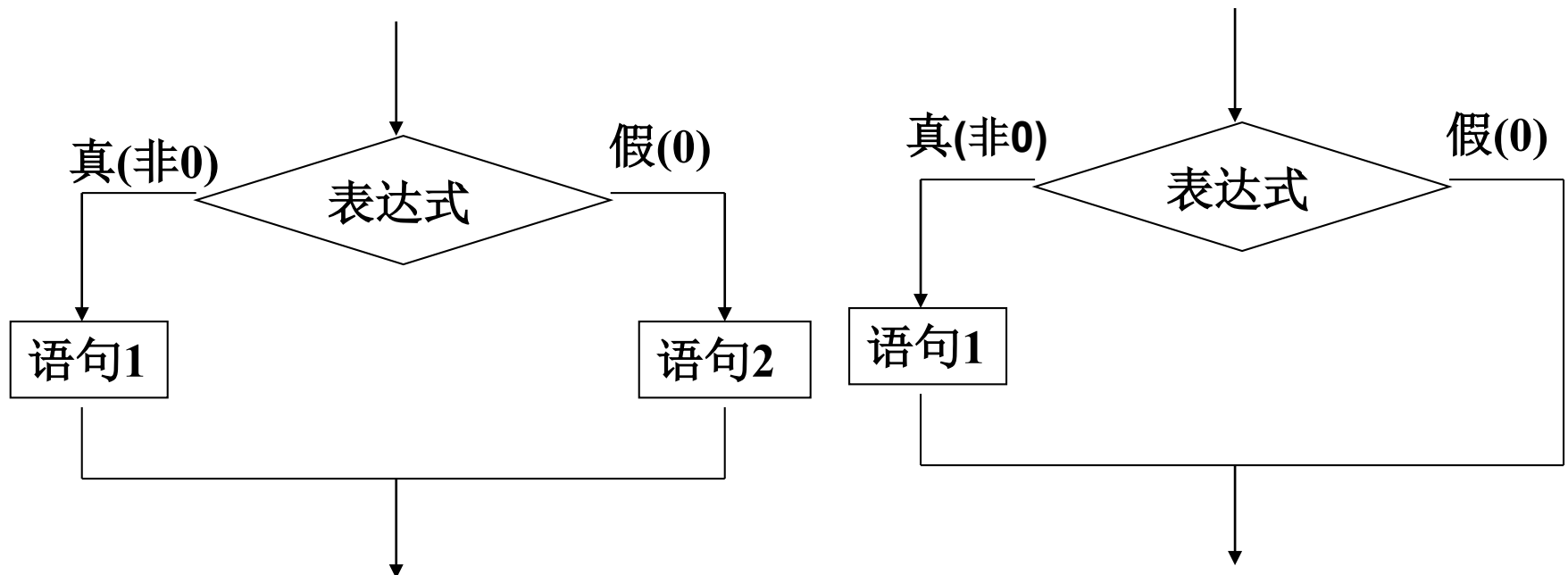
else

语句2

一条语句

if (表达式)

语句1



❖ if – else 或不带 else

判断数字的奇偶性

例3-2

输入1个整数，判断该数是奇数还是偶数。

读入一个整数

if (该数能被2整除)

则该数为偶数

else

该数为奇数

```
number % 2 == 0
```

源程序-判断数字的奇偶性

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

```
int main (void)
```

```
{ int number;
```

```
printf ("Enter a number: ");
```

```
scanf ("%d", &number);
```

```
if (number % 2 == 0){
```

```
printf ("The number is even. \n");
```

```
}
```

```
else{
```

```
printf("The number is odd. \n");
```

```
}
```

```
return 0;
```

```
}
```

Enter a number: **1028**
The number is even.

Enter a number: **329**
The number is odd.

统计学生的成绩

例3-3 输入一个正整数 n ，再输入 n 个学生的成绩，计算平均分，并统计不及格成绩的个数。

```
for (i = 1; i <= n; i++){  
    输入1个学生的成绩 grade  
    累加成绩 total  
    统计不及格成绩的个数 count  
}
```

输出结果

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

```
int main (void)
```

```
{ int count, i, n;
```

```
double grade, total;
```

```
printf ("Enter n: "); scanf ("%d", &n);
```

```
total = 0; count = 0;
```

```
for (i = 1; i <= n; i++){
```

```
    printf ("Enter grade #%d: ", i);
```

```
    scanf ("%lf", &grade);
```

```
    total = total + grade;
```

```
    if (grade < 60){
```

```
        count++;}
```

此处省略else

```
}
```

```
printf ("Grade average = %.2f\n", total/n);
```

```
printf ("Number of failures = %d\n", count);
```

```
return 0;
```

```
}
```

源程序-统计成绩

Enter n: 4

Enter grade #1: 67

Enter grade #2: 54

Enter grade #3: 88

Enter grade #4: 73

Grade average = 70.50

Number of failures = 1

3.1.3 多分支结构和else – if 语句

else-if 语句是最常用的实现多分支（多路选择）的方法

```
if (表达式1)
    语句1;
else if (表达式2)
    语句2;
.....
else if (表达式n-1)
    语句n-1;
else
    语句n;
```

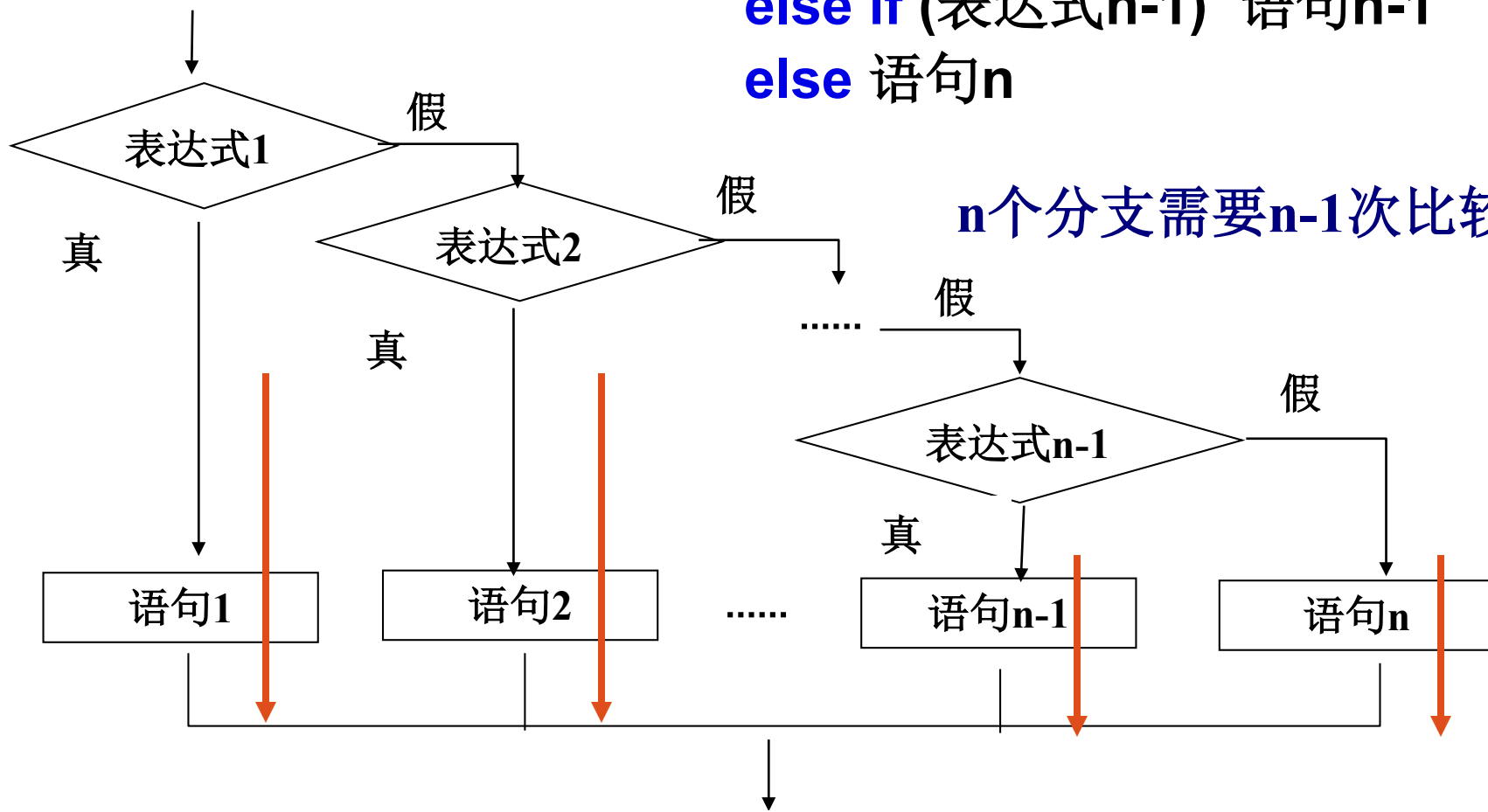
else – if 语句

if (表达式1) 语句1
else if (表达式2) 语句2

.....

else if (表达式n-1) 语句n-1
else 语句n

n个分支需要n-1次比较



改写例3-1中的判断过程，用else-if实现

```
if (yournumber == mynumber){  
    printf ("Ok! you are right!\n");  
else{  
    if (yournumber > mynumber ){  
        printf ("Sorry! your number is bigger than my number!\n");  
    else{  
        printf ("Sorry! your number is smaller than my number!\n");  
    }  
}
```

```
if (yournumber == mynumber)  
    printf ("Ok! you are right!\n");  
else if (yournumber > mynumber )  
    printf ("Sorry! your number is bigger than my number!\n");  
else  
    printf ("Sorry! your number is smaller than my number!\n");
```


更改例2-4中的分段计算水费的问题

例3-4 例2-4中提出的分段计算水费的问题。居民应交水费 y (元) 与月用水量 x (吨) 的函数关系式修正如下，并编程实现。

$$y = f(x) = \begin{cases} 0, & x < 0 \\ \frac{4x}{3}, & 0 \leq x \leq 15 \\ 2.5x - 10.5, & x > 15 \end{cases}$$

源程序-分段计算水费

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

```
int main (void)
```

```
{
```

```
    double x, y;
```

```
    printf ("Enter x:");
```

```
    scanf ("%lf", &x);
```

```
    if (x < 0){
```

```
        y = 0;
```

```
    }
```

```
    else if (x <= 15){
```

```
        y = 4 * x / 3;
```

```
    }
```

```
    else{
```

```
        y = 2.5 * x - 10.5;
```

```
    }
```

```
    printf ("f(%.2f) = %.2f\n", x, y);
```

```
    return 0;
```

```
}
```

$$y = f(x) = \begin{cases} 0 & x < 0 \\ \frac{4x}{3} & 0 \leq x \leq 15 \\ 2.5x - 10.5 & x > 15 \end{cases}$$

Enter x: **-0.5**

f(-0.50) = 0.00

Enter x: **?**

Enter x: **9.5**

f(9.50) = 12.67

Enter x: **?**

Enter x: **21.3**

f(21.30) = 42.75

3.2 四则运算

输入一个形式如“操作数 运算符 操作数”的四则运算表达式，输出运算结果。

3.2.1 程序解析

3.2.2 字符类型

3.2.3 字符型数据的输入和输出

3.2.4 逻辑运算

3.2.1 程序解析

例3-5 输入一个形式如“操作数 运算符 操作数”的四则运算表达式，输出运算结果。

输入：3.1+4.8

输出：=7.90

输入：5.0-3.9

输出：=1.1

value1 **op** value2

op: 存放一个字符 **+** **-** ***** **/** 等

if(**op** == **'+'**) value1 + value2

else if (**op** == **'-'**) value1 – value2

else if (**op** == **'*'**) value1 * value2

else if (**op** == **'/'**) value1 / value2

else "Unknown operator"

源程序-四则运算

Type in an expression: **3.1+4.8**
=7.90

```
#include <stdio.h>
int main (void)
{ double value1, value2;
  char op;
  printf ("Type in an expression: ");
  scanf ("%lf%c%lf", &value1, &op, &value2);
  if(op == '+'){
    printf ("=%.2f\n", value1 + value2); }
  else if (op == '-'){
    printf ("=%.2f\n", value1 - value2);}
  else if (op == '*'){
    printf ("=%.2f\n", value1 * value2);}
  else if (op == '/'){
    printf ("=%.2f\n", value1 / value2);}
  else{
    printf ("Unknown operator\n");}
  return 0;
}
```

3.2.2 字符型数据

```
char op; /* 定义字符型变量 */
```

char: 类型名
op: 变量名

```
scanf ("%lf%c%lf", &value1, &op, &value2);  
if (op == '+'){  
    .....  
}  
else if (op == '-'){  
    .....  
}  
else if (op == '*'){  
    .....  
}  
else if (op == '/'){  
    .....  
}  
else{  
    .....  
}
```

字符型数据

- 字符常量: '+' '-' '*' '/'
- 字符变量: op

字符常量

'a' 'z' 'A' 'Z' '0' '9' ' ' '\n'

ASCII字符集：列出所有可用的字符

每个字符：惟一的次序值（**ASCII 码**）

ASCII 码表

'0'-'9'

'A'-'Z'

'a'-'z'

区分数字1
和数字字符'1'

符 号	10进制	符 号	10进制	符 号	10进制	符 号	10进制
@	64	P	80	,	96	p	112
A	65	Q	81	a	97	q	113
B	66	R	82	b	98	r	114
C	67	S	83	c	99	s	115
D	68	T	84	d	100	t	116
E	69	U	85	e	101	u	117
F	70	V	86	f	102	v	118
G	71	W	87	g	103	w	119
H	72	X	88	h	104	x	120
I	73	Y	89	i	105	y	121
J	74	Z	90	j	106	z	122
K	75	[91	k	107	{	123
L	76	\	92	l	108		124
M	77]	93	m	109	}	125
N	78	^	94	n	110	~	126
O	79	-	95	o	111	☐	127

字符变量

char op;

定义字符型变量**op**，用于存放字符型数据。

op = '+';

将字符型常量 '+'
赋值给字符型变量op

char op1, op2;

op1 = 'A';

op2 = op1;

3.2.3 字符型数据的输入和输出

■ 调用scanf和printf输入输出字符

double value1, value2;

char operator;

printf ("Type in an expression: ");

scanf ("%lf%c%lf", &value1, &op, &value2);

printf ("%0.2f %c %0.2f", value1, op, value2);

Type in an expression: 10.0+5.61

10.00 + 5.61

输入时，操作数和运算符
之间不能出现空格

■ 字符输入函数 `getchar ()`

输入一个字符

```
char ch;
```

```
ch = getchar ();
```

■ 字符输出函数 `putchar ()`

输出一个字符

```
putchar (输出参数);
```

字符常量或字符变量



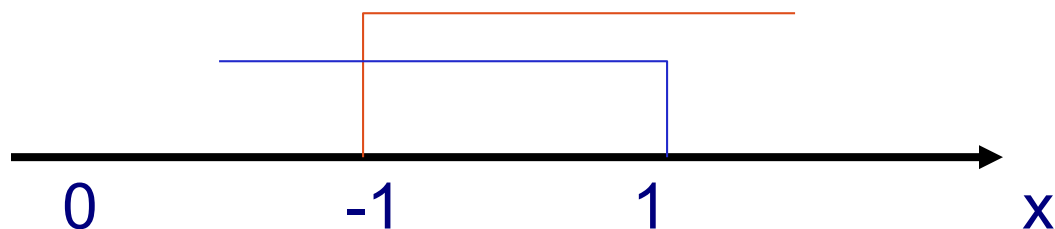
```
char ch;  
ch = getchar ();  
putchar (ch);  
putchar ('?');
```

a
a?

只能处理单个字符的输入输出
即调用一次函数，只能输入或
者输出一个字符

3.2.4 逻辑运算

$-1 \leq x \leq 1$



$x \geq -1$ 并且 $x \leq 1$

$x \geq -1$ **&&** $x \leq 1$

判断英文字母:

```
char ch;
```

```
printf ("Enter a character: ");
```

```
ch = getchar ();
```

```
if ((ch >= 'a' && ch <= 'z') || (ch >= 'A' && ch <= 'Z'))
```

```
    printf ("It is a letter.\n");
```

```
else
```

```
    printf ("It is not a letter.\n");
```

Enter a character: **d**
It is a letter.

Enter a character: **?**
It is not a letter.

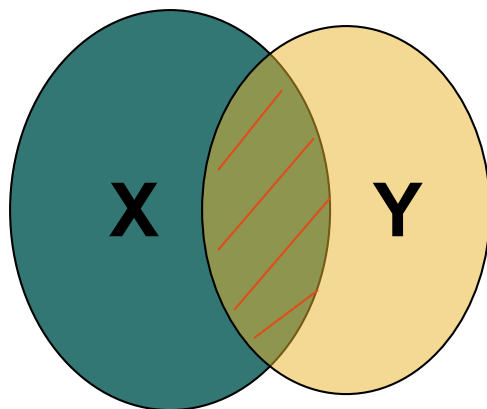
3种逻辑运算符

逻辑与 **&&**

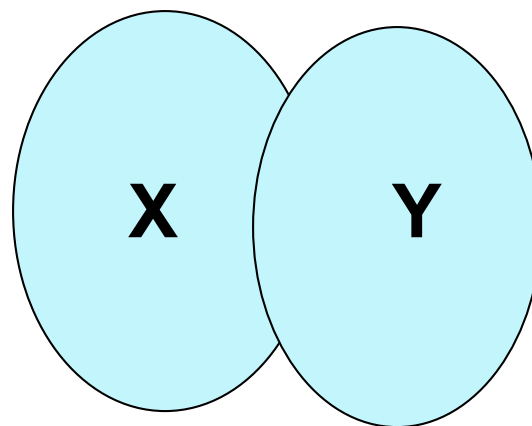
逻辑或 **||**

逻辑非 **!**

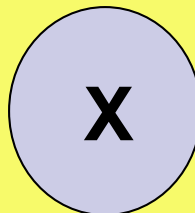
X && Y



X || Y



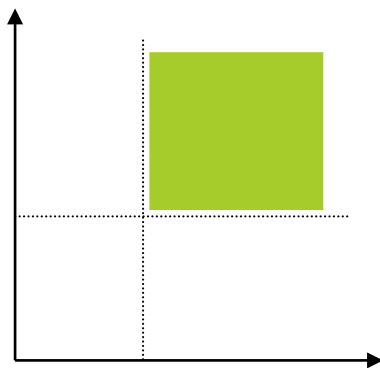
! X



逻辑运算符的含义

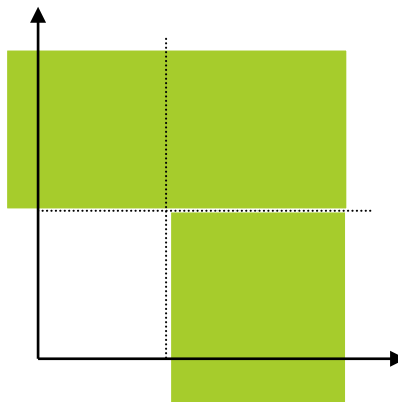
逻辑与 **&&**

$(x>1)\&\&(y>1)$



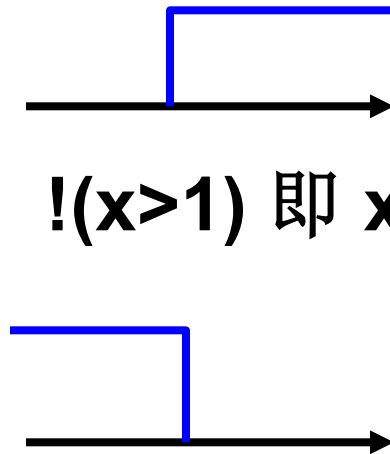
逻辑或 **||**

$(x>1)\|\|(y>1)$



逻辑非 **!**

$(x>1)$



!(x>1) 即 $x \leq 1$

逻辑运算符的功能

逻辑与 **&&** 逻辑或 **||** 逻辑非 **!**

a b		a&&b		a b	!a
假	假	假	假	真	
假	真	假	真	真	
真	假	假	真	假	
真	真	真	真	假	

逻辑表达式

逻辑表达式:

用逻辑运算符将逻辑运算对象连接起来的式子。

(ch >= 'a') && (ch <= 'z')

判断ch 是否为小写英文字母

或:

ch >= 'a' && ch <= 'z'

(ch >= 'a' && ch <= 'z') || (ch >= 'A' && ch <= 'Z')

判断ch 是否为英文字母，分大小写

条件的表示

例3-6 写出满足下列条件的C表达式。

- **ch** 是空格或者回车

`ch == ' ' || ch == '\n'`

- **ch** 是数字字符

`ch >= '0' && ch <= '9'`

- **number**是偶数

`number % 2 == 0`

- **year** 是闰年，即 **year** 能被 4 整除但不能被 100 整除，或 **year** 能被 400 整除

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

分类统计字符

例3-7 输入10个字符，统计其中英文字母、数字字符和其他字符的个数。

```
for (i = 1; i <= 10; i++){  
    输入1个字符 ch  
    if (ch是英文字母) letter++;  
    else if(ch是数字字符) digit++;  
    else other++;  
}
```

输出分类统计结果

源程序-统计字符

```
# include <stdio.h>
int main (void)
{  int digit, i, letter, other;
   char ch;
   digit = letter = other = 0;
   printf ("Enter 10 characters: ");
   for (i = 1; i <= 10; i++){
       ch = getchar();    /* 从键盘输入一个字符，赋值给变量 ch */
       if ((ch >= 'a' && ch <= 'z' ) || ( ch >= 'A' && ch <= 'Z'))
           letter ++;
       else if (ch >= '0' && ch <= '9') /* 如果ch是数字字符 */
           digit ++;
       else
           other ++;
   }
   printf ("letter=%d, digit=%d, other=%d\n", letter, digit, other);
   return 0;
}
```

input 10 characters: **Reold 123?**
letter=5, digit=3, other=2

3.3 查询自动售货机中商品的价格

例3-8 查询自动售货机中商品的价格

3.3.1 程序解析

3.3.2 **switch**语句

3.3.3 多分支结构

3.3.1 程序解析

例3-8 自动售货机出售4种商品：

薯片(crisps)、爆米花(popcorn)、巧克力(chocolate)和可乐 cola)，

对应价格：3.0、2.5、4.0

在屏幕上显示菜单：

- [1] Select crisps
- [2] Select popcorn
- [3] Select chocolate
- [4] Select cola
- [0] Exit

显示菜单

输入选项choice

if (choice == 0) 退出

choice==1~4 → price赋单价

choice==其他 → price赋0

输出单价

}

约定： 可以连续不超过5次查询商品的价格，超过5次时自动退出；不到5次时，用户可以选择退出。

输入输出： 当用户输入编号1~4，显示相应商品的价格；输入0，退出查询；输入其他编号，显示价格为0。

```

#include <stdio.h>
int main (void)
{
    int choice, i;    double price;
    for( i = 1; i <= 5; i++) {
        printf ("[1] Select crisps \n");
        printf ("[2] Select popcorn \n");
        printf ("[3] Select chocolate \n");
        printf ("[4] Select cola \n");
        printf ("[0] exit \n");
        printf ("Enter choice: ");
        scanf ("%d", &choice);
        if (choice == 0) break;
        switch (choice) {
            case 1: price=3.0; break;
            case 2: price=2.5; break;
            case 3: price=4.0; break;
            case 4: price=3.5; break;
            default: price=0.0; break;
        }
        printf ("price = %0.1f\n", price);
    }
    printf ("Thanks \n");
}

```

```

[1] Select crisps
[2] Select popcorn
[3] Select chocolate
[4] Select cola
[0] Exit
Enter choice: 1
price = 3.0
[1] Select crisps
[2] Select popcorn
[3] Select chocolate
[4] Select cola
[0] Exit
Enter choice: 7
price = 0.0
[1] Select crisps
[2] Select popcorn
[3] Select chocolate
[4] Select cola
[0] Exit
Enter choice: 0
Thanks

```

3.3.2 switch语句

处理多分支选择问题，3种情况

1. 在switch语句的每个语句段中都使用break语句

```
switch(表达式){  
    case 常量表达式1: 语句段1; break;  
    case 常量表达式2: 语句段2; break;  
    .....  
    case 常量表达式n: 语句段n ; break;  
    default :          语句段n+1; break;  
}
```

```

switch(表达式){
    case 常量表达式1: 语句段1; break;
    case 常量表达式2: 语句段2; break;
        .....
    case 常量表达式n: 语句段n; break;
    default :          语句段n+1; break;
}

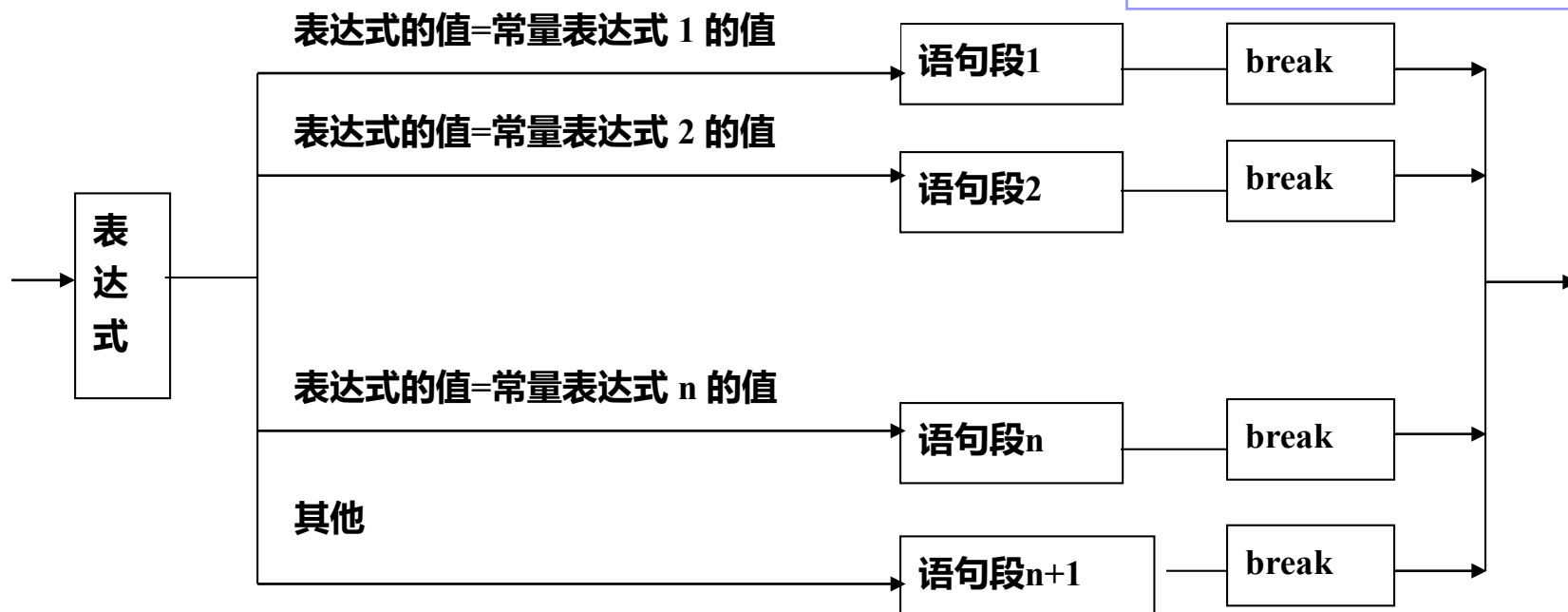
```

```

switch (choice) {
    case 1: price=3.0; break;
    case 2: price=2.5; break;
    case 3: price=4.0; break;
    case 4: price=3.5; break;
    default: price=0.0; break;
}

```

用else-if 如何实现？



求解简单表达式

例3-9 输入一个形式如“操作数 运算符 操作数”的四则运算表达式，输出运算结果。
(要求用switch语句实现)

value1 op value2

op: 存放一个字符 + - * / 等

switch (op){

case '+': value1 + value2

case '-': value1 - value2

case '*': value1 * value2

case '/': value1 / value2

default: "Unknown operator"

}

案例---

输入: 3.1+4.8

输出: 7.9

源程序

```
# include <stdio.h>
int main (void)
{ char op; double value1, value2;
  printf ("Type in an expression: ");
  scanf ("%lf%c%lf", &value1, &op, &value2);
  switch (op){
    case '+':
      printf ("=%.2f\n", value1+value2); break;
    case '-':
      printf ("=%.2f\n", value1-value2); break;
    case '*':
      printf ("=%.2f\n", value1*value2); break;
    case '/':
      printf ("=%.2f\n", value1/value2); break;
    default:
      printf ("Unknown operator\n"); break;
  }
  return 0;
}
```

错误:
case op=='+'

如果除数为0?

Type in an expression: 3.1+4.8
=7.9

2. 在switch中不使用break

```
switch(表达式){  
    case 常量表达式1: 语句段1;  
    case 常量表达式2: 语句段2;  
    .....  
    case 常量表达式n: 语句段n;  
    default :          语句段n+1;  
}
```

switch (表达式){

case 常量表达式1: 语句段1;

case 常量表达式2: 语句段2;

.....

case 常量表达式n: 语句段n;

default : 语句段n+1;

}

switch (choice) {

case 1: price=3.0;

case 2: price=2.5;

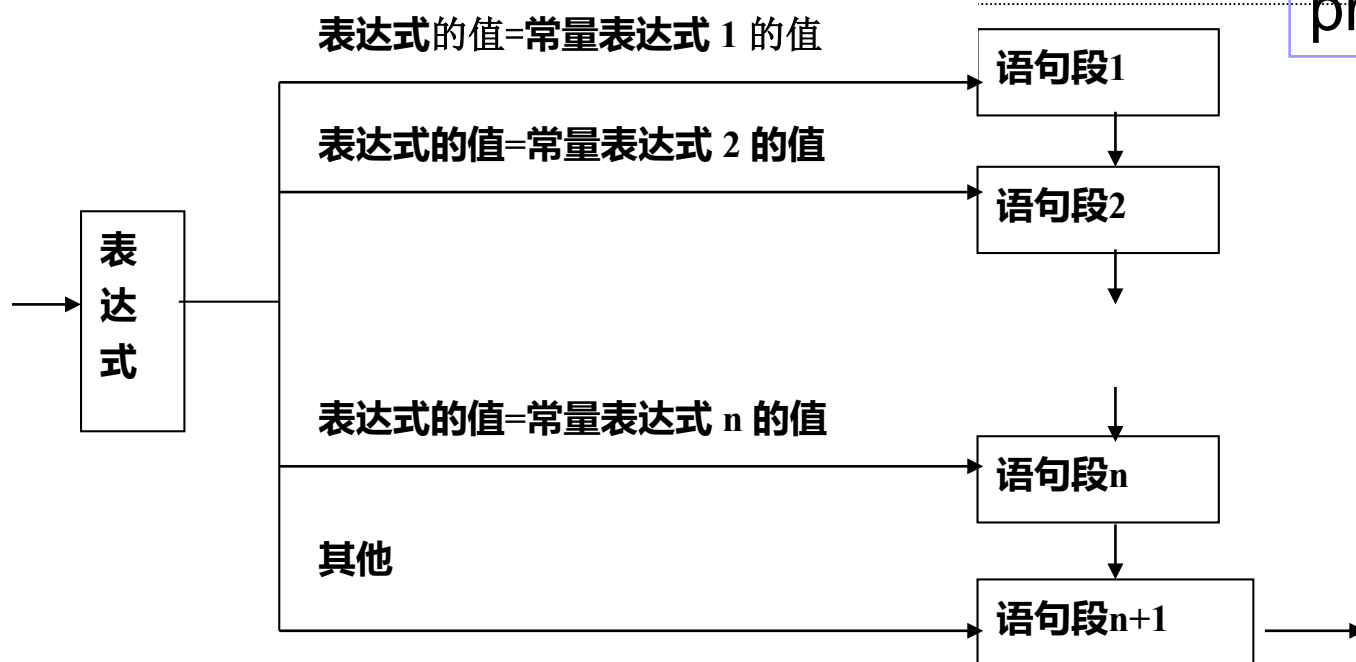
case 3: price=4.0;

case 4: price=3.5;

default: price=0.0;

}

price=?



3. 在switch的某些语句段中使用break

例3-10统计字符类别。

输入**10**个字符，分别统计3种类别字符：
(1) 空格或回车 (2) 数字字符 (3) 其他字符。

比较：例3-7 输入**10**个字符，统计其中
(1) 英文字母 (2) 数字字符 (3) 其他字符

```

int main (void)
{
    int blank, digit, i, other; char ch;
    blank = digit = other = 0;
    printf ("Enter 10 characters: ");
    for (i = 1; i <= 10; i++){
        ch = getchar ();
        switch (ch){
            case ' ':
            case '\n':
                blank ++; break;
            case '0' : case '1' : case '2' : case '3' : case '4' :
            case '5' : case '6' : case '7' : case '8' : case '9' :
                digit ++; break;
            default:
                other ++; break;
        }
    }
    printf ("blank=%d, digit=%d, other=%d\n", blank, digit, other);
    return 0;
}

```

Enter 10 characters: **Reold 123?**
blank=1, digit=3, other=6

3.3.3 多分支结构

- 分支结构一般分为二分支和多分支两种结构
- 二分支结构用基本的 **if** 语句实现
- 多分支结构用实现方法：
 - **else – if** 语句
 - **switch**语句
 - 嵌套的 **if - else**语句

嵌套的 if-else 语句

if (表达式)

语句1 ← **if** 语句

else

语句2 ← **if** 语句

if (表达式1)

if (表达式2) 语句1

else 语句2

else

if (表达式3) 语句3

else 语句4

例3-1 简单的猜数游戏

```
if (yournumber == mynumber){  
    printf ("Ok! you are right!\n");  
}else{  
    if (yournumber > mynumber ){  
        printf ("Sorry! your number is bigger than my number!\n");  
    }else{  
        printf ("Sorry! your number is smaller than my number!\n");  
    }  
}
```

if (表达式1)

嵌套的 if – else 语句

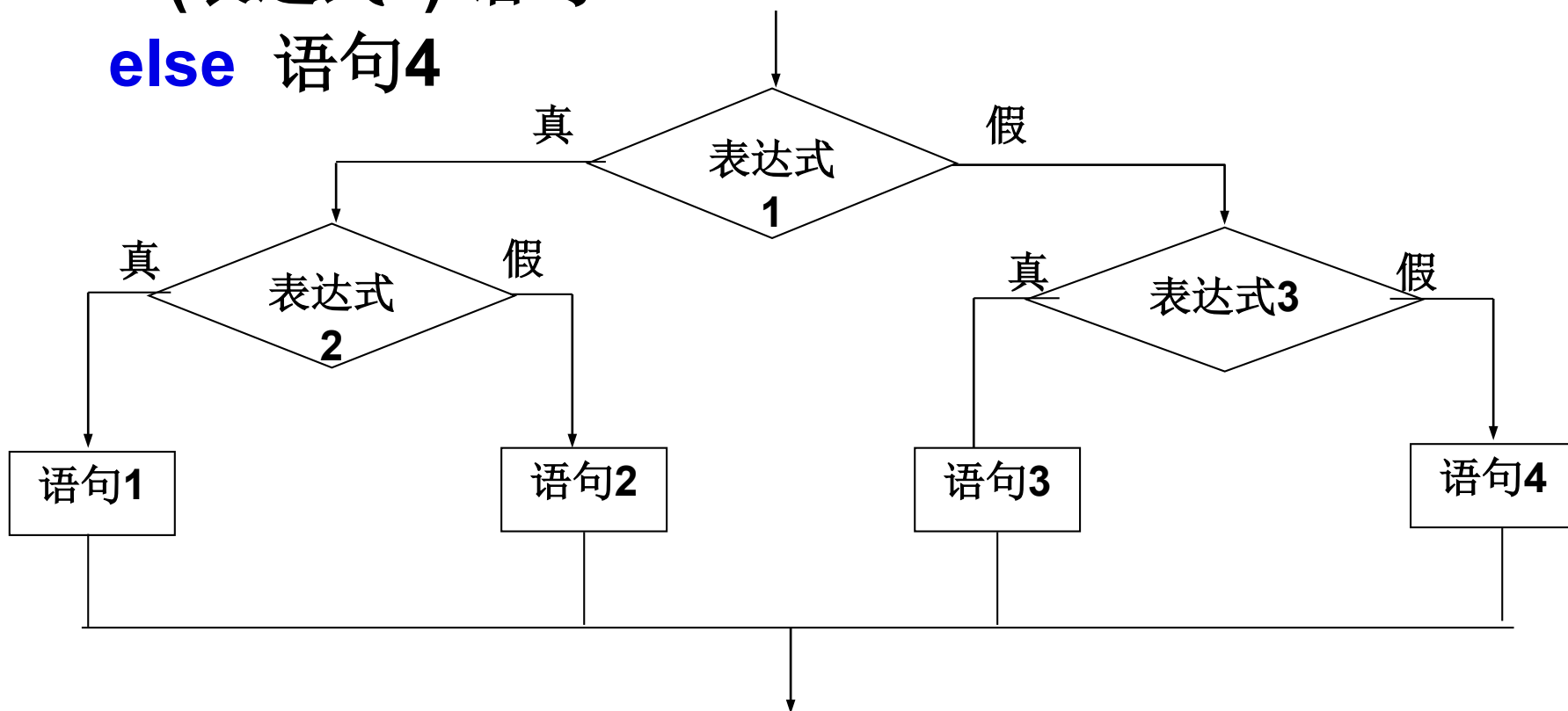
if (表达式2) 语句1

else 语句2

else

if (表达式3) 语句3

else 语句4




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

```
int main (void)
```

```
{ double value1, value2;
```

```
char op;
```

```
printf ("Type in an expression: ");
```

```
scanf ("%lf%c%lf", &value1, &op, &value2);
```

```
if (op == '+')
```

```
    printf("=%.2f\n", value1 + value2);
```

```
else if (op == '-')
```

```
    printf ("=%.2f\n", value1 - value2);
```

```
else if (op == '*')
```

```
    printf ("=%.2f\n", value1 * value2);
```

```
else if (op == '/')
```

```
    if (value2 != 0)
```

/* 嵌套的if, 判断除数是否为0 */

```
        printf ("=%.2f\n", value1 / value2);
```

```
    else
```

```
        printf ("Divisor can not be 0!\n");
```

```
else
```

```
    printf ("Unknown operator!\n");
```

```
return 0;
```

```
}
```

例3-11 改进例3-5，求解简单表达式。要求对除数为0的情况作特别处理。

Type in an expression: **3.1+4.8**
=7.9

Type in an expression: **3.4/0**
Divisor can not be 0!

else 和 if 的匹配

```
if (表达式1)
    if (表达式2) 语句1
    else 语句2
else
    if (表达式3) 语句3
    else 语句4
```

```
if (表达式1)
    if (表达式2) 语句1

else
    if (表达式3) 语句3
    else 语句4
```

else 与最靠近它的、没有与别的 else 匹配过的 if 匹配

```
if (表达式1)
    if (表达式2) 语句1
    else
        if (表达式3) 语句3
        else 语句4
```

改变else 和 if 的配对

例3-12 改写下列 if 语句，使 else 和第1个 if 配对。

if (x < 2)

if (x < 1) y = x + 1;

else y = x + 2;

每条语句的执行条件？

if (x < 2){

if (x < 1) y = x + 1;

}

else y = x + 2;

if (x < 2)

if (x < 1) y = x + 1;

else;

else y = x + 2;

本章总结

■ 分支结构

- **if-else**语句
- **else if**/嵌套的**if-else**

■ **switch**语句

- **case**后为常量表达式
- **break**的使用

■ 数据类型：**char**型

■ 运算符与表达式

- 逻辑运算符、关系运算符
- 逻辑表达式

- 正确理解**if**语句和**switch**语句的执行机制；
- 掌握各类关系表达式、逻辑表达式的运用；
- 能运用分支语句熟练编写分支结构类的程序。

第四章 循环 要点

- 循环次数未明示（**while**）
- 循环结束控制的三种基本方式
- 至少循环一次（**do-while**）
- 循环多出口控制（**break**和**continue**）
- 循环结束后如何判断哪个出口？
- 嵌套循环及其初始化
- 三种循环控制的选择

第四章 循环结构

4.1 用格雷戈里公式求 π 的近似值 (**while**语句)

4.2 统计一个整数的位数 (**do-while**语句)

4.3 判断素数 (**break** 和 **continue** 语句)

4.4 求 $1! + 2! + \dots + 100!$ (循环嵌套)

4.5 循环结构程序设计

4.1 用格里高利公式求 π 的近似值

例4-1 使用格雷戈里公式求 π 的近似值，要求精确到最后一项的绝对值小于 10^{-4} 。

$$\frac{\pi}{4} = 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \square$$

4.1.1 程序解析

4.1.2 while语句

4.1.1 程序解析—求 π 的近似值

例4-1 使用格雷戈里公式求 π 的近似值，要求精确到最后一项的绝对值小于 10^{-4} 。

$$\frac{\pi}{4} = 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \square$$

例2-8 求 $1 - 1/3 + 1/5 - \dots$ 的前 n 项和

```
flag = 1;
```

```
denominator = 1;
```

```
item = 1;
```

```
sum = 0;
```

```
while (fabs(item) >= 0.0001)
```

```
for (i = 1; i <= n; i++) {
```

```
    sum = sum + item;
```

```
    flag = -flag;
```

```
    denominator = denominator + 2;
```

```
    item = flag * 1.0 / denominator;
```

```
}
```

循环结束条件:

$|item| < 0.0001$

循环条件:

$|item| \geq 0.0001$

$\text{fabs}(item) \geq 0.0001$

❖ 循环次数未知

例4-1源程序—求 π 的近似值

```
# include <math.h>
int main (void)
```

pi = 3.141613

```
{
```

```
    int denominator, flag;
```

item = 0.0 ?

```
    double item, pi;
```

```
    flag = 1; denominator = 1; item = 1.0; pi = 0;
```

```
    while (fabs (item) >= 0.0001) {
```

```
        pi = pi + item;
```

fabs (item) < 0.0001?

```
        flag = -flag;
```

```
        denominator = denominator + 2;
```

```
        item = flag * 1.0 / denominator;
```

```
    }
```

```
    pi = pi + item;
```

```
    pi = pi * 4;
```

```
    printf ("pi = %f\n", pi);
```

```
    return 0;
```

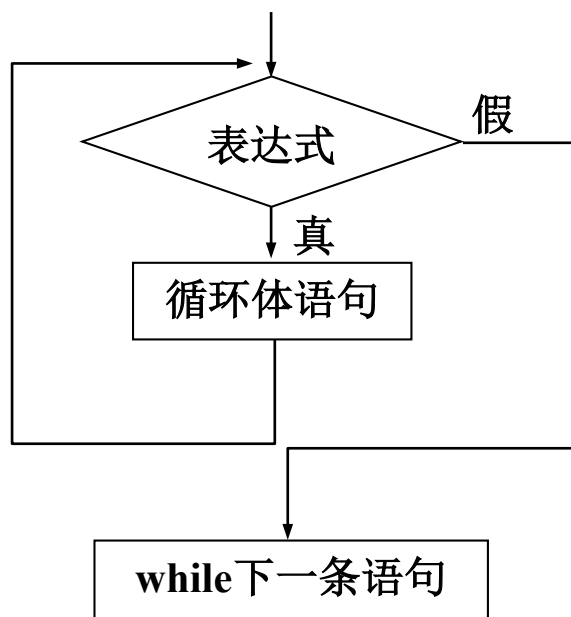
```
}
```

❖ 循环次数未知

4.1.2 while 语句

while (条件)
循环体语句;

一条语句



循环条件

循环体

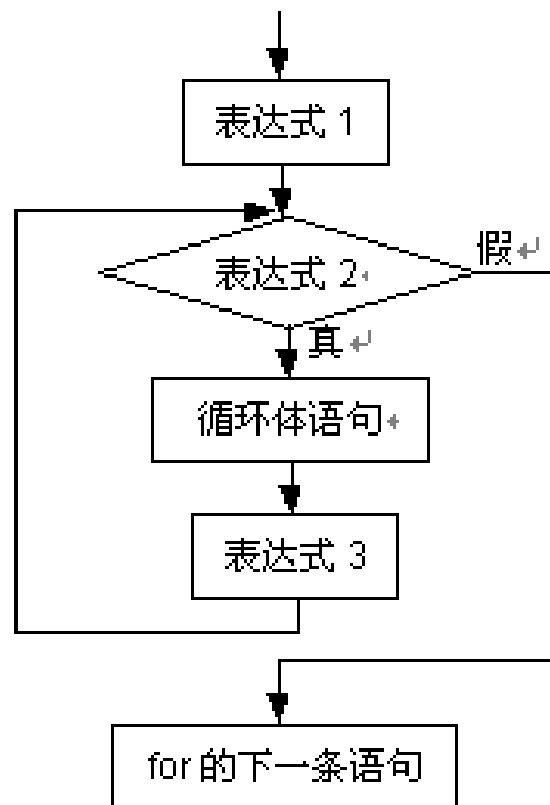


图 4-2 for 语句的执行流程

while 语句说明

while 语句和 for 语句

都是在循环前先判断条件

把 for 语句改写成 while 语句

for (表达式1; 表达式2; 表达式3)

循环体语句

表达式1;

```
while (表达式2) {  
    for的循环体语句;  
    表达式3;  
}
```

while 和 for 的比较

```
for (i = 1; i <= 10; i++){  
    sum = sum + i;  
}
```

```
    i = 1;                                /* 循环变量赋初值 */  
    while (i <= 10){                       /* 循环条件 */  
循环体      sum = sum + i;  
              i++;                         /* 循环变量的改变 */  
    }
```

统计学生的成绩

例4-2 从键盘输入一批学生的成绩，计算平均成绩，并统计不及格学生的人数。

例3-3 输入一个正整数n，再输入n个学生的成绩，计算平均分，并统计不及格成绩的个数。

```
total = 0;
```

```
count = 0;
```

```
for (i = 1; i <= n; i++){  
    scanf ("%lf", &grade);  
    total = total + grade;  
    if (grade < 60)  
        count++;  
}
```

如何确定
循环条件？

分析-统计成绩

从键盘输入一批学生的成绩，计算平均成绩，并统计不及格学生的人数。

■ 确定循环条件

- 不知道输入数据的个数，无法事先确定循环次数；
- 用一个特殊的数据作为正常输入数据的结束标志，比如选用一个负数作为结束标志。

循环结束条件： **grade < 0**

循环条件： **grade >= 0**

程序段-统计成绩

```
total = 0;  
count = 0;
```

```
for (i = 1; i <= n; i++){  
    scanf ("%lf", &grade);  
    total = total + grade;  
  
    if (grade < 60){  
        count++;    }  
}
```

```
num = 0;  
total = 0;  
count = 0;  
scanf ("%lf", &grade);  
while (grade >= 0) {  
    scanf ("%lf", &grade);  
    total = total + grade;  
    num ++;  
    if (grade < 60){  
        count++;    }  
    scanf ("%lf", &grade);  
}
```

❖ 循环结束条件

例4-2 源程序-统计成绩

```
#include <stdio.h>
int main (void)
{   int count, num; double grade, total;
    num = 0; total = 0; count=0;
    printf ("Enter grades: \n");
    scanf ("%lf", &grade);    /* 输入第1个数*/
    while (grade >= 0) {      /* 输入负数, 循环结束 */
        total = total + grade;
        num++;
        if (grade < 60) count++;
        scanf ("%lf", &grade);
    }
    if(num != 0) {
        printf("Grade average is %.2f\n", total/num);
        printf("Number of failures is %d\n", count);
    }else
        printf("Grade average is 0\n");
    return 0;
}
```

Enter grades: 67 88 73 54 82 -1
Grade average is 72.80
Number of failures is 1

小结-常见的 循环控制方式

■ 计数控制的循环 第2.4节

```
sum = 0;  
for (i = 1; i <= n; i++){  
    sum = sum + i;  
}
```

■ 计算值控制的循环 例4-1

```
item = 1.0;  
while (fabs (item) >= 0.0001) {  
    item = flag * 1.0 / denominator;  
    .....
```

■ 输入值控制的循环 伪数据 例4-2

```
scanf ("%lf", &grade);  
while (grade >= 0) {  
    .....
```

```
scanf ("%lf", &grade);  
}
```

❖ 循环结束控制方式

4.2 统计一个整数的位数

例4-3 从键盘读入一个整数，统计该数的位数。

4.2.1 程序解析

4.2.2 do - while语句

4.2.1 程序解析-统计一个整数的位数

例4-3 从键盘读入一个整数，统计该数的位数。

分析：以**495**为例，自右向左，一位一位地数：

- 从个位数开始数；
第1次数出：**5** ($495 \% 10$)
- 数好一位，就划掉该数字，得到一个新数**49**；**49~~5~~**
舍去**5**，得到新数**49** ($495 / 10 = 49$)
- 继续从新数的个位数开始数；
第2次数出：**9** ($49 \% 10$)
- 数好一位，就划去该数字，再得到一个**新数4**；**4~~95~~**
舍去**9**，得到新数**4** ($49 / 10 = 4$)
- 继续从新数的个位数开始数；
第3次数出：**4** ($4 \% 10$)
- 数好一位，就划去该数字，再得到一个**新数0**；**4~~95~~**
舍去**4**，得到新数**0** ($4 / 10 = 0$)
- 全部数好，全部划去，遇零结束。共重复**3**次，**3**位数。

算法-统计一个整数的位数

number = 0?

digit = numbr % 10

number = number / 10

number = 495

495 % 10

count = 0;

number = 49

49 % 10

do{

number = 4

4 % 10

digit = number % 10

number = 0 结束

number = number / 10;

■ 循环结束条件

number == 0

count ++;

■ 循环条件

number != 0

} while (number != 0);

■ 循环体

□ digit = number % 10

□ number = number / 10

□ count ++

count = 0;

while (number != 0){

digit = number % 10

number = number / 10;

count ++;

}

❖ do-while 循环次数

例4-3 源程序-统计整数位数

```
int main (void)
{
    int count, number;
    printf ("Enter a number: ");
    scanf ("%d", &number) ;
    if (number < 0) number = -number;
    count = 0;
    do {
        number = number / 10;
        count ++;
    } while (number != 0);
    printf ("It contains %d digits.\n", count);
    return 0;
}
```

```
count = 0;
if (number == 0) count = 1;
while (number != 0) {
    number = number / 10;
    count ++;
}
```

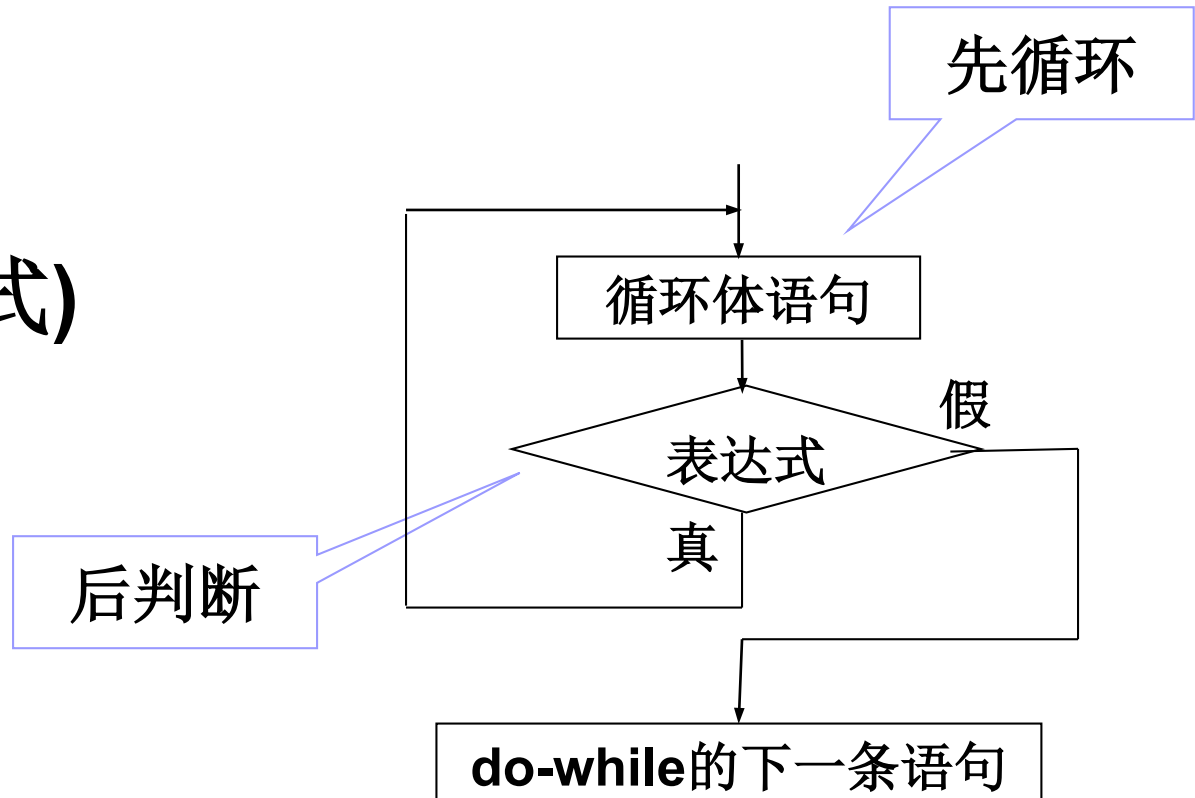
Enter a number: **12534**
It contains 5 digits.

Enter a number: **-99**
It contains 2 digits.

Enter a number: **0**
It contains 1 digits.

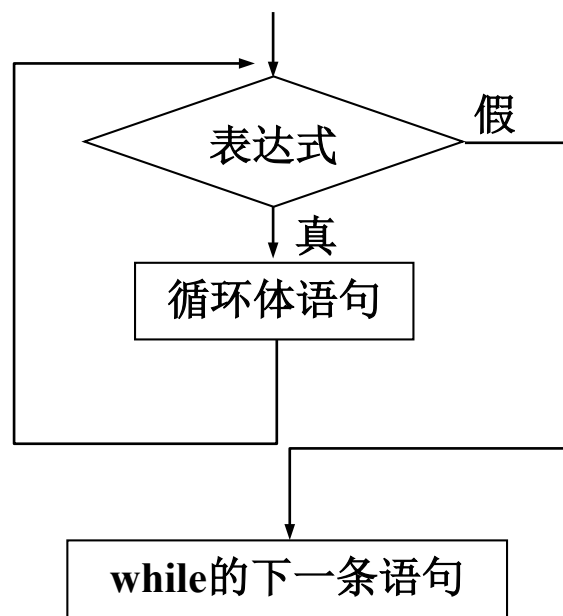
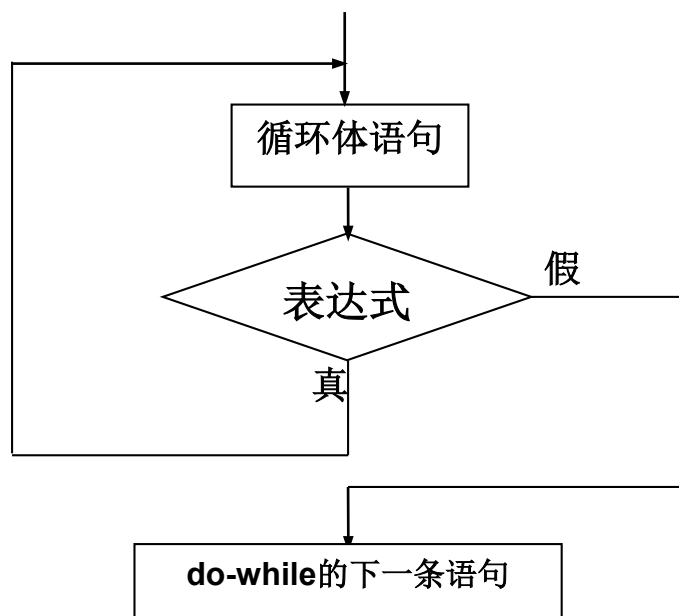
4.2.2 do - while 语句

do {
 循环体语句
} **while** (表达式)



while 和 do-while 的比较

- **while**: 先判别条件，再决定是否循环；
- **do-while**: 先至少循环一次，然后再根据条件决定是否继续循环。



4.3 判断素数

例4-4 输入一个正整数 m ，判断它是否为素数。素数就是只能被1和自身整除的正整数，1不是素数，2是素数。

4.3.1 程序解析

4.3.2 **break** 语句 和 **continue** 语句

4.3.1 程序解析—判断素数

算法：除了 1 和 m ，不能被其它数整除。

```
for (i = 2; i <= m/2; i++)  
    if (m % i == 0) break;  
if (i > m/2) printf ("yes\n")  
else printf ("no\n");
```

m	$\%2$	$\%3$	$\%4$	$\%5$	$\%(m-1)$
不是素数	\parallel	$=0$	$=0$		
是素数	$\&\&$	$\neq 0$	$\neq 0$		

m 不可能被大于 $m/2$ 的数整除

i 取值 $[2, m-1]$ 、 $[2, m/2]$ 、 $[2, \sqrt{m}]$

❖ 提前结束循环

例4-4 源程序1-判断素数

```
int main (void)
```

```
{ int i, m;
```

```
    printf ("Enter a number: ");
```

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

```
    for (i = 2; i <= m/2; i++){
```

```
        if (m % i == 0)
```

```
            break;
```

```
    }
```

```
    if (i > m/2 && m != 1) /* 1不是素数 */
```

```
        printf ("%d is a prime number! \n", m);
```

```
    else
```

```
        printf ("No!\n");
```

```
}
```

Enter a number: 9

No!

Enter a number: 11

11 is a prime number!

Enter a number: 1

No!

Enter a number: 2

2 is a prime number!

循环条件?

循环的结束条件?

4.3.2 break 语句

```
while ( exp ) {
```

语句1

```
if ( expb )
```

```
break;
```

语句2

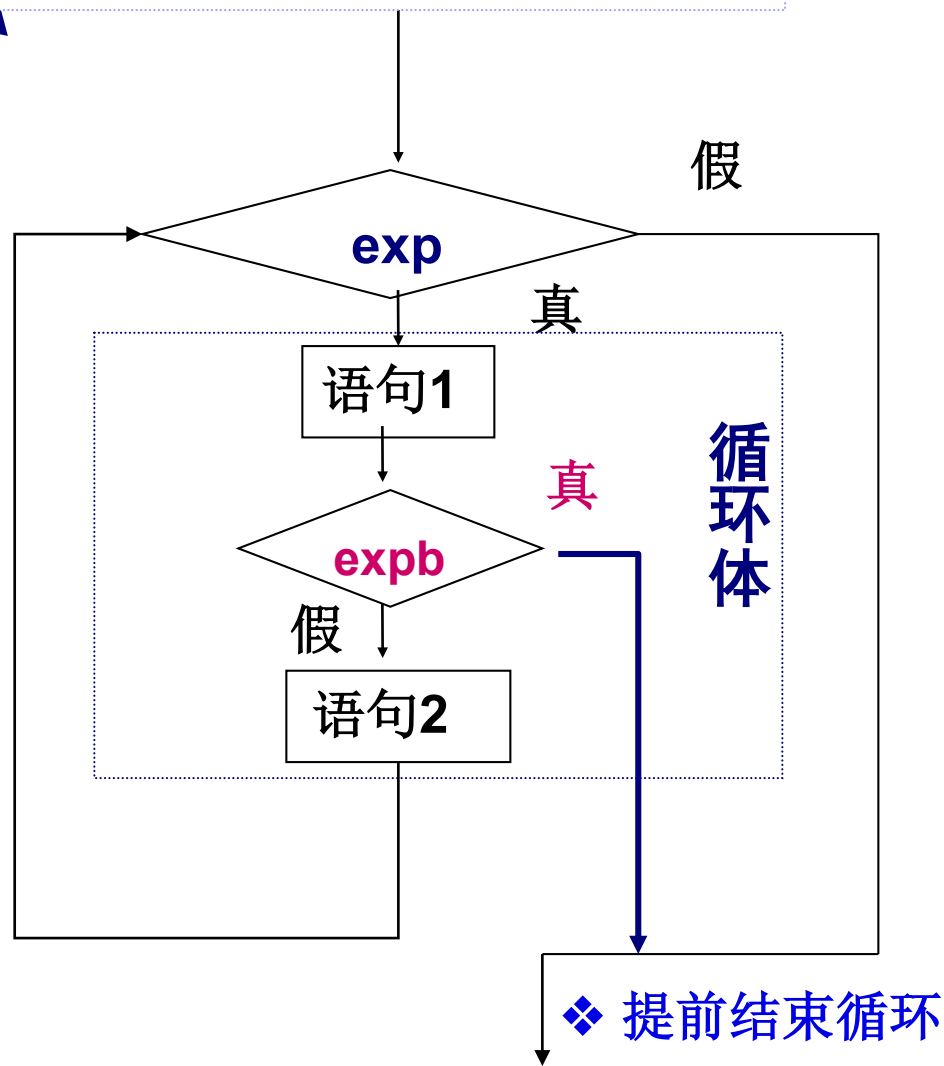
```
}
```

当循环有**多个出口**时:

- 共同表示循环条件
- 区分结束条件

```
for ( i = 2; i <= m/2; i++ )  
    if ( m % i == 0 ) break;
```

```
if ( i > m/2 ) printf("Yes!");  
else printf("No!\n");
```



练习-输出结果是什么？

Enter a number: 9
No!

```
for (i = 2; i <= m/2; i++)  
    if (m%i == 0){  
        printf("No!\n");  
        break;  
    }  
printf("Yes!");
```

```
for (i = 2; i <= m/2; i++)  
    if (m % i == 0) break;  
if (i > m/2 ) printf("Yes!");  
else printf("No!\n");
```

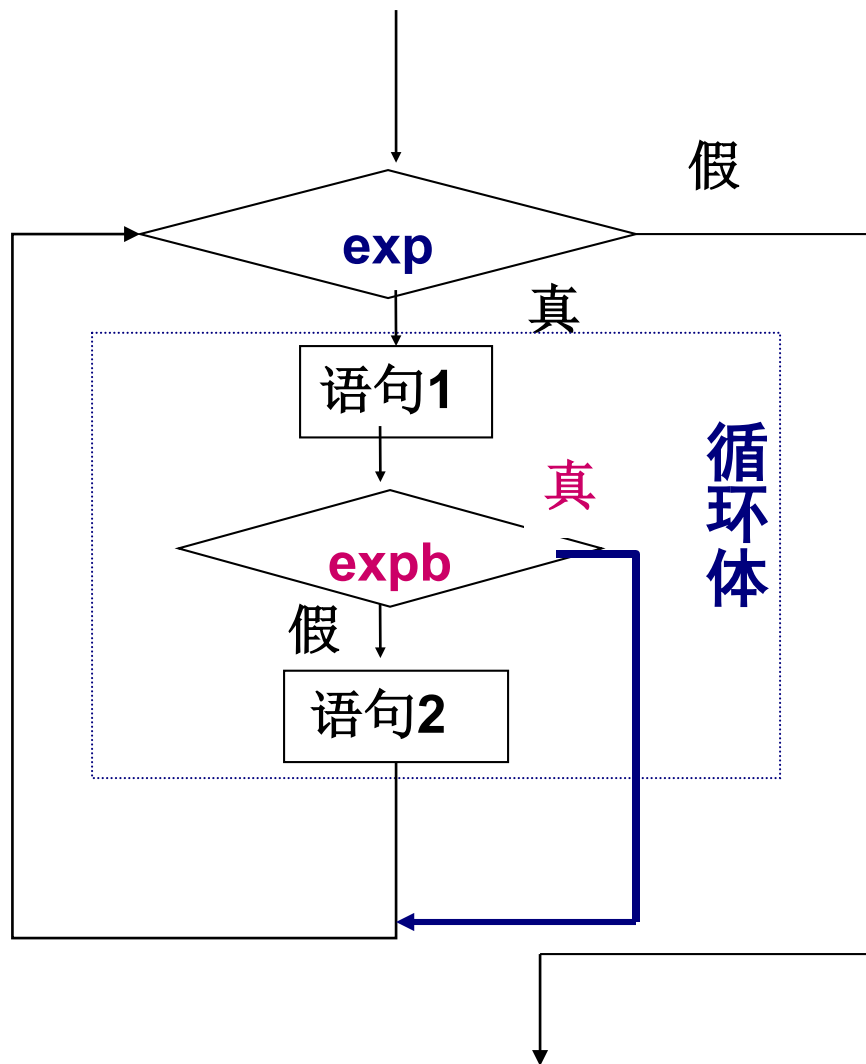
Enter a number: 9
No!
Yes!

```
for (i = 2; i <= m/2; i++)  
    if (m % i == 0)  
        printf ("No!\n");  
    else  
        printf ("Yes! \n");
```

Enter a number: 9
Yes!
No!
Yes!

continue 语句

```
while (exp){  
    语句1  
    if (expb)  
        continue;  
    语句2  
}
```



跳过**continue**后面的语句，继续下一次循环

break和continue

```
# include <stdio.h>
int main (void)
{ char c;
  int i;
  for (i = 0; i < 10; i++) {
    c = getchar ();
    if (c == '\n') continue;
    putchar (c);
  }
}
```

abc ✓
efgh ✓
123 ✓

abc

abcefg1

break和continue

输出100~200之间所有能被3整除的数

```
for (i = 100; i <= 200; i++) {  
    if ( i % 3 != 0 )  
        continue;  
    printf ("%d ", i);  
}
```

```
for (i = 100; i <= 200; i++)  
    if (i % 3 == 0)  
        printf ("%d ", i);
```

例4-5-1 简单的猜数游戏，最多允许猜7次。

```
int main (void)
{   int count = 0, flag, mynumber=38, yournumber;
    flag = 0;  /* flag: 为0表示没猜中，为1表示猜中了*/
    for (count = 1; count <= 7; count++) {
        scanf ("%d", &yournumber);
        if (yournumber == mynumber) {
            printf ("Lucky You!\n");
            flag = 1;
            break;          /* 已猜中，游戏结束，终止循环 */
        } else if (yournumber > mynumber )
            printf ("Too big\n");
        else printf ("Too small\n");
    }
    if (flag == 0)    printf ("Game Over!\n");
    return 0;
}
```

❖ 设置flag---标记循环出口

例4-4源程序2-判断素数

```
int main (void)
```

```
{ int i, m;
```

```
printf ("Enter a number: ");
```

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

```
for (i = 2; i <= m/2; i++){
```

```
    if (m % i == 0)
```

```
        break;
```

```
}
```

```
if (i > m/2 && m != 1)
```

```
    printf ("Yes\n");
```

```
else
```

```
    printf ("No\n");
```

```
}
```

```
int main (void)
```

```
{ int i, flag=1, m;
```

```
printf ("Enter a number: ");
```

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

```
if (m == 1) flag = 0; /* 改为非素数*/
```

```
for (i = 2; i <= m/2; i++) {
```

```
    if (m % i == 0){
```

```
        flag = 0; break;
```

```
    }
```

```
}
```

```
if (flag == 1)
```

```
    printf ("Yes\n");
```

```
else
```

```
    printf ("No\n");
```

```
}
```

❖ 判断循环如何结束

例4-5-2 简单的猜数游戏，最多允许猜7次。

```
# include <stdlib.h>
# include <time.h>
int main (void)
{ int count = 0, flag = 0, mynumber, yournumber;
  srand (time(0));
  mynumber = rand () % 100 + 1; /* 随机产生一个1~100之间的被猜数 */
  while (count < 7){
    printf ("Enter your number: "); scanf ("%d", &yournumber);
    count++;
    if (yournumber == mynumber) {
      printf ("Lucky You!\n"); flag = 1; break;
    }
    else if (yournumber > mynumber ) printf ("Too big\n");
    else printf ("Too small\n");
  }
  if (flag == 0) printf ("Game Over!\n");
  return 0;
}
```

❖ 设置flag---提前结束循环

4.4 求 $1! + 2! + \dots + 100!$

```
sum = 0;  
for (i = 1; i <= 100; i++){  
    item = i !  
    sum = sum + item;  
}
```

4.4.1 程序解析

调用函数 **fact(i)** 计算 i 的阶乘

4.4.2 嵌套循环

用循环计算 i 的阶乘

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

```
double fact (int n);
```

```
int main (void)
```

```
{   int i;
```

```
    double sum;
```

```
    sum = 0;
```

```
    for ( i = 1; i <= 100; i++ ){
```

```
        sum = sum + fact (i);
```

```
    }
```

```
    printf ("1! + 2! + 3! + ... + 100! = %e\n", sum);
```

```
    return 0;
```

```
}
```

```
double fact (int n)
```

```
{   int i;
```

```
    double result = 1;
```

```
    for ( i = 1; i <= n; i++)
```

```
        result = result * i;
```

```
    return result;
```

```
}
```

4.4.1 程序解析

求 $1! + 2! + \dots + 100!$

$$1! + 2! + \dots + 100! = 9.426900e+157$$

❖ 调用函数求n! (double类型)

4.4.2 嵌套循环

```
sum = 0;
for ( i = 1; i <= 100; i++ ){
    item = i !
    sum = sum + item;
}
```

```
sum = 0;
for ( i = 1; i <= 100; i++ ) {
    item = 1;
    for (j = 1; j <= i; j++){
        item = item * j;
    }
    sum = sum + item;
}
```

例4-7 源程序

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

```
int main(void)
```

```
{  int i, j;
```

```
    double item, sum;
```

```
    sum = 0;
```

```
    for( i = 1; i <= 100; i++ ) {
```

```
        item = 1;
```

```
        for ( j = 1; j <= i; j++ ){
```

```
            item = item * j;
```

```
        }
```

```
        sum = sum + item;
```

```
    }
```

```
    printf("1! + 2! + 3! + ... + 100! = %e\n", sum);
```

```
}
```

/* item 存放阶乘 */

/* 每次求阶乘都从1开始 */

/* 内层循环算出 item = i! */

讨论-内层循环的初始化

求 $1! + 2! + \dots + 100!$

```
sum = 0;
```

```
for ( i = 1; i <= 100; i++ ) {
```

```
    item = 1;
```

```
    for (j = 1; j <= i; j++){  
        item = item * j;
```

```
    }
```

```
    sum = sum + item;
```

```
}
```

```
item = 1;
```

```
sum = 0;
```

```
for( i = 1; i <= 100; i++ ){
```

```
    for (j = 1; j <= i; j++){  
        item = item * j;
```

```
    }
```

```
    sum = sum + item;
```

```
}
```

求 $1! + 1! * 2! + \dots + 1! * 2! * \dots * 100!$

❖ 循环嵌套的初始化位置

分析嵌套循环的执行过程

```
sum = 0;
```

```
for(i = 1; i <= 100; i++) {
```

```
    item = 1;
```

```
    for (j = 1; j <= i; j++)
```

```
        item = item * j;
```

```
    sum = sum + item;
```

```
}
```

n 外层循环变量 **i** 的每个值

内层循环变量 **j** 变化一个轮次;

n 内外层循环变量的名字不能相同

分别用 **i** 和 **j**

练习-运行结果是什么？

```
for (i = 1; i <= 100; i++)  
    for (j = 1; j <= i; j++)  
        printf ("%d %d\n", i, j );
```

i = 1	j = 1	输出 1 1 （第 1 次输出）
i = 2	j = 1	输出 2 1 （第 2 次输出）
	j = 2	输出 2 2 （第 3 次输出）
.....		
i = 100	j = 1	输出 100 1 （第 4951 次输出）
	j = 2	输出 100 2 （第 4952 次输出）
	
	j = 100	输出 100 100 （第 5050 次输出）

例4-12古典算术问题—搬砖头

某地需要搬运砖块，已知男人一人搬3块，女人一人搬2块，小孩两人搬一块。
问用45人正好搬45块砖，有多少种搬法？

```
for ( men = 0; men <= 45; men++ )  
    for ( women = 0; women <= 45; women++ )  
        for ( child = 0; child <= 45; child++ )  
            if ((men+women+child==45) && (men*3+women*2+child*0.5==45))  
                printf("men=%d women=%d child=%d\n", men, women, child);  
}
```

例4-12 源程序(2)

```
for (men = 0; men <= 15; men++)  
    for (women = 0; women <= 22; women++){  
        child = 45 - women - men;  
        if (men * 3 + women * 2 + child * 0.5 == 45)  
            printf("men=%d women=%d child=%d\n", men, women, child);  
    }
```

比较循环次数

```
for (men = 0; men <= 45; men++)  
    for (women = 0; women <= 45; women++)  
        for (child = 0; child <= 45; child++)  
            if ((men+women+child==45) && (men*3+women*2+child*0.5==45))  
                printf("men=%d women=%d child=%d\n", men, women, child);  
}
```

4.5 循环结构程序设计

■ 循环程序的实现要点

- 归纳出哪些操作需要反复执行？ 循环体
- 这些操作在什么情况下重复执行？ 循环条件

■ 常见的循环控制方式

- 计数控制、计算值控制、输入值控制
- 多重控制（计数控制+计算值控制，……）

■ 选用合适的循环语句

for while do-while

常见的循环控制方式

- 计数控制： 第2.4节, 例4-6, 4-7, 4-11, 4-12

```
sum = 0;  
for (i = 1; i <= n; i++){  
    sum = sum + fact (i);  
}
```

- 计算值控制： 例4-1, 4-3, 4-9

```
item = 1.0;  
while (fabs (item) >= 0.0001) {  
    item = flag * 1.0 / denominator;  
    .....
```

- 输入值控制为主： 例4-2, 4-8

```
scanf ("%lf", &grade);  
while (grade >= 0) {  
    .....
```

```
scanf ("%lf", &grade);  
}
```

- 计数控制+计算值控制： 例4-4, 4-5, 4-10

循环语句的选择

■ 主要考虑因素-循环控制方式

- 事先给定循环次数，首选 **for**
 - 通过其他条件控制循环，考虑 **while**或 **do-while**
- if**（循环次数已知）

使用 **for** 语句

else */* 循环次数未知 */*

if (循环条件在进入循环时明确)

使用 **while** 语句

else */* 循环条件需要在循环体中明确 */*

使用 **do-while** 语句

例4-8 输入一批学生的成绩，求最高分(for)

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

```
int main(void)
```

```
{ int i, mark, max, n;
```

```
printf("Enter n: "); scanf ("%d", &n);
```

```
printf("Enter %d marks: ", n);
```

```
scanf ("%d", &mark); /* 读入第一个成绩 */
```

```
max = mark; /* 假设第一个成绩是最高分 */
```

```
for (i = 1; i < n; i++) {
```

```
    scanf ("%d", &mark);
```

```
    if (max < mark)
```

```
        max = mark;
```

```
}
```

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

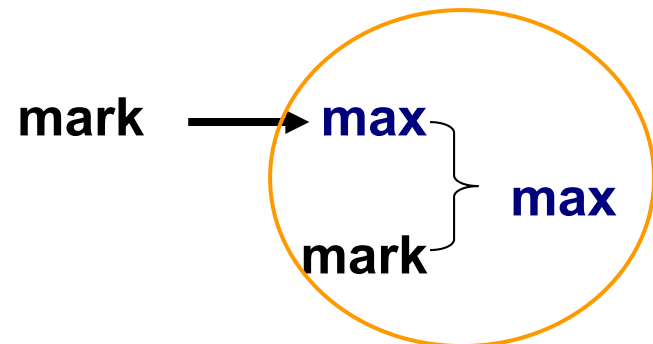
```
return 0;
```

```
}
```

Enter n: 5

Enter 5 maks: 67 88 73 54 82

Max = 88



Enter n: 0

例4-8 输入一批学生的成绩，求最高分(while)

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

```
int main(void)
```

```
{   int mark, max;
```

```
    printf("Enter marks:");
```

```
    scanf ("%d", &mark); /* 读入第一个成绩 */
```

```
    max = mark;          /* 假设第一个成绩最高分 */
```

```
    while (mark >= 0){
```

```
        if(max < mark)
```

```
            max = mark ;
```

```
        scanf ("%d", &mark );
```

```
    }
```

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

```
    return 0;
```

```
}
```

Enter marks:67 88 73 54 82 -1

Max = 88

Enter marks:-1

例4-8 输入一批学生的成绩，求最高分(do-while)

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

```
int main(void)
```

```
{ int mark, max;
```

```
max = -1; /* 给max赋一个小初值*/
```

```
printf ( "Enter marks: " );
```

```
do {
```

```
scanf ( "%d", &mark );
```

```
if ( max < mark )
```

```
max = mark;
```

```
} while ( mark >= 0 );
```

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

```
}
```

Enter marks: 67 88 73 54 82 -1

Max = 88

Enter marks: -1

例4-9 逆序问题。将一个正整数逆序输出

确定：循环条件和循环体(循环不变式)

12345 5 4 3 2 1

用do-while
实现？

12345 % 10 = 5	12345 / 10 = 1234
1234 % 10 = 4	1234 / 10 = 123
123 % 10 = 3	123 / 10 = 12
12 % 10 = 2	12 / 10 = 1
1 % 10 = 1	1 / 10 = 0 结束

循环不变式 $x \% 10$ $x = x / 10$
循环结束条件 $x == 0$

```
scanf ( "%d", &x );  
while ( x != 0 ){  
    digit = x % 10;  
    x = x / 10 ;  
    printf ( "%d ", digit );  
}
```

❖ while or do-while?

例4-10 求100以内的全部素数，每行输出10个

```
for (m = 2; m <= 100; m++)
```

```
if (m是素数) printf( "%d", m);
```

```
flag = 1;
```

```
n = sqrt(m);
```

```
for ( i = 2; i <= n; i++ )
```

```
if ( m % i == 0 ){
```

```
    flag = 0;
```

```
    break;
```

```
}
```

```
if(flag == 1) printf("yes\n");
```

```
else printf("no\n");
```

```
for ( m = 2; m <= 100; m++ ){
```

```
    flag = 1;
```

```
    n=sqrt(m);
```

```
    for(i = 2; i <= n; i++)
```

```
        if (m % i == 0){
```

```
            flag = 0;
```

```
            break;
```

```
        }
```

```
    if(flag == 1) printf("%d", m);
```

```
}
```

❖ 求一个还是多个？

```
int main(void)
```

```
{ int count, i, flag, m, n;
```

```
count = 0;
```

```
for (m = 2; m <= 100; m++){
```

```
    flag = 1;
```

```
    n = sqrt(m);
```

```
    for(i = 2; i <= n; i++){
```

```
        if (m % i == 0){
```

```
            flag = 0;
```

```
            break;
```

```
        }
```

```
    if(flag == 1){ /* 如果m是素数 */
```

```
        printf("%6d", m);
```

```
        count++;          /* 每行10个的处理 */
```

```
        if (count % 10 == 0) printf("\n");
```

```
    }
```

```
}
```

```
}
```

例4-10 源程序

2	3	5	7	11	13	17	19	23	29
31	37	41	43	47	53	59	61	67	71
73	79	83	89	97					

❖输出多个每行几个

例4-11 求Fibonacci序列： 1,1,2,3,5,8,13,...

1, 1, 2, 3, 5, 8, 13,

x1 x2 x

x1 x2 x

x1 = x2 = 1;

x = x1 + x2;

x1 = x2;

x2 = x;

```
x1 = 1;
```

```
x2 = 1;
```

```
printf ( "%6d%6d", x1, x2 ); /* 输出头两项 */
```

```
for ( i = 1; i <= 8; i++ ){ /* 循环输出后8项 */
```

```
    x = x1 + x2; /* 计算新项 */
```

```
    printf( "%6d", x );
```

```
    x1 = x2; /* 更新x1和x2 */
```

```
    x2 = x;
```

```
}
```

❖关注循环体.....

本章总结

- 循环结构以及
 - while do-wh
 - break与cont
 - 嵌套循环
 - 综合程序设计（循环结构）
 - 常用算法
 - 例题：求 π 、拆分整数、求素数、猜数、求最值、求fibonacci
 - 习题：求水仙花数、求最大公约数/最小公倍数、求ddd
- 理解 while和 do-while的执行机制；
 - 掌握 break 和 continue 的作用方式；
 - 掌握嵌套循环的执行机制与设计方法；
 - 能熟练循环语句编写循环结构类的程序；
 - 熟练掌握常用算法

本章（函数）要点

- 函数的作用？如何确定函数功能？
- 怎样定义函数？如何调用函数？定义函数与声明函数有何区别？
- 什么是函数的参数？怎样确定函数的参数？
- 在函数调用时，参数是如何传递数据的？
- 变量与函数有什么关系？如何使用局部变量和全局变量？
- 什么是静态变量？
- 变量的生命周期和作用域

Chap 5 函数

5.1 计算圆柱体积

5.2 数字金字塔

5.3 复数运算

5.1 计算圆柱体积

- 5.1.1 程序解析
- 5.1.2 函数的定义
- 5.1.3 函数的调用
- 5.1.4 函数程序设计

5.1.1 程序解析—计算圆柱体积

例5-1 输入圆柱体的高和半径，求圆柱体积，
 $\text{volume}=\pi*r^2*h$ 。

要求定义和调用函数 **cylinder (r, h)** 计算圆柱体的体积。

例5-1源程序

```
/* 计算圆柱体积 */  
# include <stdio.h>  
double cylinder (double r, double h); /* 函数声明*/  
int main( void )  
{  
    double height, radius, volume;  
  
    printf ("Enter radius and height: ");  
    scanf ("%lf%lf", &radius, &height);  
    /* 调用函数，返回值赋给volume */  
    volume = cylinder (radius, height );  
    printf ("Volume = %.3f\n", volume);  
  
    return 0;  
}
```

例5-1源程序

Enter radius and height: 3.0 10

Volume = 282.743

/* 定义求圆柱体积的函数 */

double cylinder (double r, double h)

{

double result;

result = 3.1415926 * r * r * h; /* 计算体积 */

return result; /* 返回结果 */

}

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

```
double cylinder (double r, double h); /* 函数声明 */
```

```
int main( void )
```

```
{
```

```
    double height, radius, volume;
```

```
    printf ("Enter radius and height: ");
```

```
    scanf ("%lf%lf", &radius, &height);
```

```
    volume = cylinder (radius, height);
```

```
    printf ("Volume = %.3f\n", volume);
```

```
    return 0;
```

```
}
```

```
double cylinder (double r, double h)
```

```
{
```

```
    double result;
```

```
    result = 3.1415926 * r * r * h;
```

```
    return result;
```

```
}
```

例5-1源程序

Enter radius and height: 3.0 10

Volume = 282.743

问题:

函数是如何运行的?

5.1.2 函数的定义

- 函数是指完成一个**特定工作**的独立并可以**反复使用**的**程序片段**（注意与“程序模块”的区别）。
 - 库函数：由C语言系统提供定义
如**scanf ()**、**printf ()**等函数
 - 自定义函数：需要用户自己定义
如计算圆柱体体积函数**cylinder ()**
- **main ()**也是一个函数，C程序由一个**main ()**或多个函数构成。
- 程序中一旦调用了某个函数，该函数就会**完成特定的功能**，然后返回到调用它的地方。
 - 函数经过运算，得到一个明确的运算结果，并需要回送该结果。例如，函数**cylinder ()**返回圆柱的体积。

```
double cylinder (double r, double h)
{ double result;
  result = 3.1415926 * r * r * h;
  return result;
}
```

5.1.2 函数定义

函数返回值的类型

函数类型 函数名 (形参表)

没有分号

/* 函数首部 */

/* 函数体 */

函数实现过程

return 表达式;

只能返回一个值

把函数运算的结果回送给主函数

❖ 函数的原型：确定数据传递方法

分析函数的定义

函数类型

函数名

形参表

↓ ↓ ↓

```
double cylinder (double r, double h)    /* 函数首部 */  
{                                         /* 函数体，写在一对大括号内 */
```

```
    double result;
```

```
    result = 3.1415926 * r * r * h;      /* 计算圆柱体积 */
```

```
    return result;                       /* 返回运算结果 */
```


```
}
```

↑
与函数类型一致

❖ 函数的定义的原型：确定调用方法

形参

不能写成 `double r, h`



```
double cylinder (double r, double h)
{ double result;
  result = 3.1415926 * r * r * h;
  return result;
}
```

函数类型 函数名 (形参表)

```
{
  函数实现过程
  return 表达式;
}
```



类型1 参数1 ， 类型2 参数2 ，, 类型n 参数n

参数之间用逗号分隔，每个参数前面的类型都必须分别写明

5.1.3 函数的调用

- 定义一个函数后，就可以通过程序来调用这个函数。
- 调用标准库函数时，在程序的最前面用 **#include** 命令包含相应的头文件。
- 调用自定义函数时，程序中必须有与调用函数相对应的函数定义。

1. 函数调用的形式

- 函数调用的一般形式为：

函数名 (实际参数表)

- 对于实现计算功能的函数，函数调用通常出现在三种（教材上说两种）情况：

- 赋值语句调用

volume = *cylinder* (radius, height);

- 表达式调用（含输出函数的实参）

printf (“%f”, *cylinder* (radius, height));

- 语句调用（放弃使用函数返回值，或void类型）

printf (“%f”, radius);

2. 函数调用的过程

- 计算机在执行程序时，从主函数**main**开始执行，如果遇到某个函数调用，主函数被暂停执行，转而执行相应的函数，该函数执行完后，将返回主函数，然后再从原先暂停的位置继续执行。
- 函数遇**return**返回主函数

分析函数调用的过程

```
# include <stdio.h>
double cylinder (double r, double h);
int main( void )
{   double height, radius, volume;
    printf ("Enter radius and height: ");
    scanf ("%lf%lf", &radius, &height);
    volume = cylinder (radius, height );
    printf ("Volume = %.3f\n", volume);
    return 0;
}
double cylinder (double r, double h)
{   double result;
    result =3.1415926 * r * r * h;
    return result;
}
```

调用函数

实参→形参

执行函数中的语句

返回调用它的地方

3. 参数传递

- 函数定义时的参数被称为形式参数（简称形参）

`double cylinder (double r, double h);`

- 函数调用时的参数被称为实际参数（简称实参）

`volume = cylinder (radius, height);`

- 参数传递：实参→形参 单向传递

- 在参数传递过程中，实参把值复制给形参。

- 形参和实参一一对应：数量一致，类型一致，顺序一致
（有例外：printf, scanf）

- 形参：变量，用于接受实参传递过来的值

- 实参：常量、变量或表达式

4. 函数结果返回

- 完成确定的功能（运算），将运算结果返回给主调函数。
- 函数结果返回的形式：
 - **return** 表达式;
 - **return** (表达式);
 - **return** ; /* void 类型函数的返回 */

【例5-2】定义判断奇偶数的函数even (n)

定义一个判断奇偶数的函数**even (n)**，当**n**为偶数时返回**1**，否则返回**0**。

如何调用该函数？

```
/* 判断奇偶数的函数 */
int even (int n)      /* 函数首部 */
{
    if(n%2 == 0){      /* 判别奇偶数 */
        return 1;      /* 偶数返回1 */
    }else{
        return 0;      /* 奇数返回0 */
    }
}
```

```
# include <stdio.h>
int main ( void )
{ int x,sum=0 ;
  . . . .
  if (even(x)!=0)
      sum=sum+x ;
  printf ("%d", sum);
  return 0;
}
```

❖ 函数返回结果

5. 函数原型声明

只写函数定义中的第1行（函数首部），并以分号结束。

函数类型 函数名(参数表);

`double cylinder (double r, double h);`

`void pyramid (int n);`

- 函数必须先定义后调用，将被调函数（定义）放在主调函数的前面，就像变量先定义后使用一样。
- 如果自定义函数在主调函数的后面，就需要在函数调用前，加上函数原型声明。
- 函数声明：说明函数的类型和参数的情况，以保证程序编译时能判断对该函数的调用是否正确。

要调用函数，
必须先要声明！

在执行函数调用时，实参把值计算出来，拷贝给相应位置的形参；函数执行完后，通过return ()，可返回一个结果值。

实参与形参
个数相同、类型一致

有多个实参时
后面的先计算

形参的改变，
不影响实参变量的值

只能返回一个结果，
类型与函数定义时一致

5.1.4 函数程序设计

例5-3 输入精度 ϵ ，使用格里高利公式求 π 的近似值，精确到最后一项的绝对值小于 ϵ 。要求定义和调用函数 **funpi(ϵ)** 求 π 的近似值。

$$\frac{\pi}{4} = 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \dots$$



什么做参数？

例5-3 源程序

/* 用格里高利公式计算 π 的近似值，精度为e */

```
# include <stdio.h>
# include <math.h>
double funpi (double e);
int main (void)
{   double e, pi;

    printf ("Enter e:");
    scanf ("%lf", &e);
    pi = funpi (e);
    printf ("pi = %f\n", pi);

    return 0;
}
```

Enter e: 0.0001

pi = 3.1418

```
double funpi (double e)
{   int denominator, flag;
    double item, sum;
    flag = 1;
    denominator = 1;
    item = 1.0;
    sum = 0;
    while (fabs (item) >= e){
        sum = sum + item;
        flag = -flag;
        denominator = denominator + 2;
        item = flag * 1.0 / denominator;
    }
    sum = sum + item;
    return sum * 4;
}
```

❖ 求 π 的代码

例5-4 判断素数的函数

例5-4 求100以内的全部素数，每行输出10个。素数就是只能被1和自身整除的正整数，1不是素数，2是素数。

要求定义和调用函数**prime (m)**判断m是否为素数，当m为素数时返回1，否则返回0。

- 算法描述：对2~100之间的每个数进行判断，若是素数，则输出该数。

```
for (m = 2; m <= 100; m++)
```

```
    if (m是素数)
```

```
        printf("%d ", m);
```

```
        prime (m) != 0
```

例5-4 源程序

```
#include <stdio.h>
#include <math.h>
int prime (int m);
int main(void)
{   int count, m;

    count = 0;
    for(m = 2; m <= 100; m++){
        if ( prime(m) != 0 ){
            printf("%6d", m );
            count++;
            if (count %10 == 0)
                printf ("\n");
        }
    }
    printf ("\n");
}
```

```
int prime (int m)
{   int i, n;
    if ( m <= 1 ) return 0;
    else if ( m == 2 ) return 1;
    else{
        limit = sqrt (m) + 1;
        for( i = 2; i <= limit; i++){
            if (m % i == 0){
                return 0;
            }
        }
        return 1;
    }
}
```

5.2 数字金字塔

- 5.2.1 程序解析
- 5.2.2 不返回结果的函数
- 5.2.3 结构化程序设计思想

例5-5

/* 输出数字金

#include <std

void pyramid

int main (voi

{

pyramid(5);

return 0;

}

void pyramid (int n)

{

int i, j;

for (i = 1; i <= n; i++){

for (j = 1; j <= n-i; j++){

printf(" ");

for (j = 1; j <= i; j++){

printf(" %d ", i);

putchar ('\n');

}

}

```
for (i = 1; i <= n; i++) {  
    for (j = 1; j <= n-i; j++)  
        printf(" ");  
    一行中的数字显示
```

```
for (i = 1; i <= n; i++) {  
    一行中的空格处理;  
    一行中的数字显示  
}
```

/* 调用函数，输出数字金字塔 */

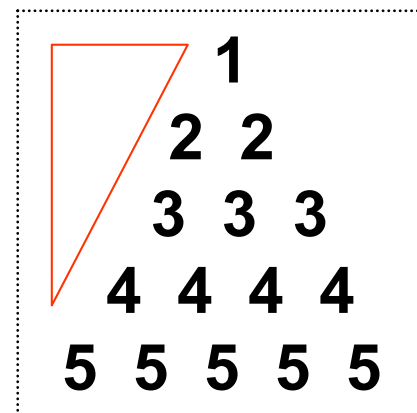
/* 函数定义 */

/* 需要输出的行数 */

/* 输出每行左边的空格 */

/* 输出每行的数字 */

/* 每个数字的前后各有一个空格 */



5.2.2 不返回运算结果的函数定义

表示不返回结果

↓

```
void 函数名（参数表）    /* 函数首部 */  
{                          /* 函数体 */  
    函数实现过程  
    return;              /* 若是函数的最后语句，可以省略return */  
}
```

这类函数通常用于屏幕输出等

不能省略, 否则
函数类型被默认定义为int

5.2.2 不返回运算结果的函数定义

- 由于函数没有返回结果，函数调用不可能出现在表达式中，通常以**独立的调用语句方式**，如`pyramid(5);`;
- 不返回结果的函数，在定义、调用、参数传递、函数声明上，思路完全与以前相同，只是**函数类型变为void**。
- 它适用把一些确定的、相对独立的**程序功能包装成函数**。
 - 主函数通过调用不同的函数，体现算法步骤
 - 各步骤的实现由相应函数完成
 - **简化主调函数结构**，以体现结构化程序设计思想。

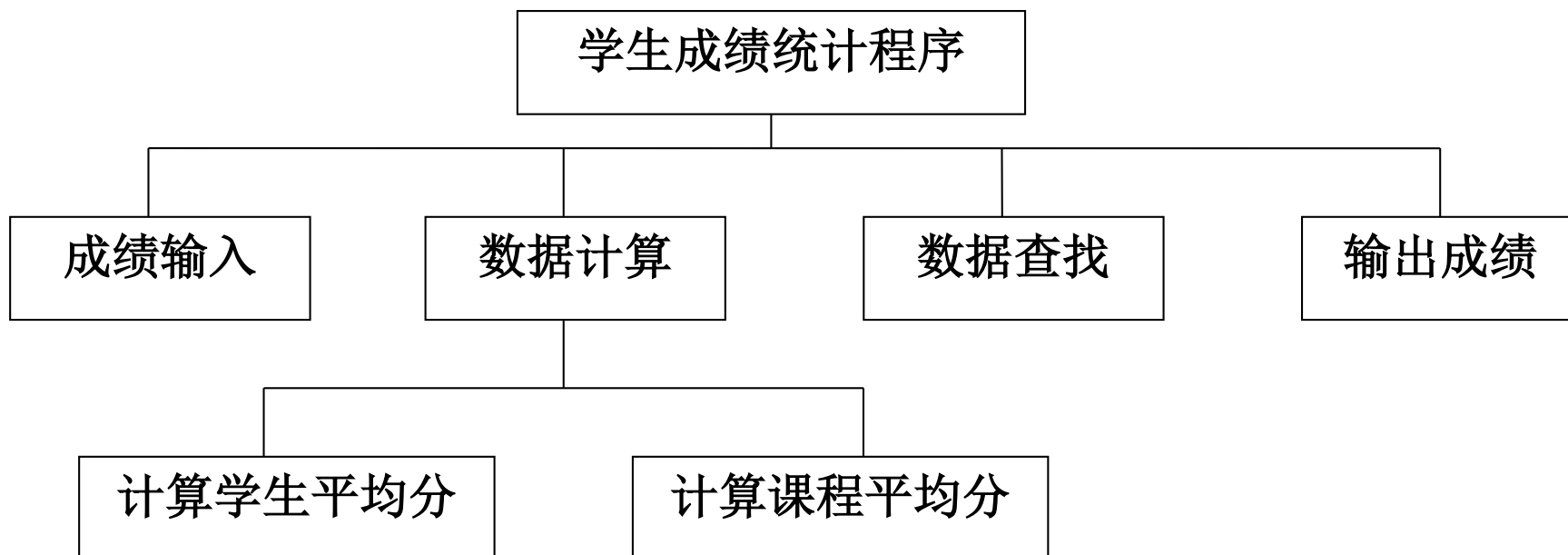
5.2.3 结构化程序设计思想

- 结构化程序设计(**Structured Programming**)
 - 程序设计技术
 - C语言是结构化程序设计语言
- 强调程序设计的风格和程序结构的规范化, 提倡清晰的结构
 - 基本思路是将一个复杂问题的求解过程划分为若干阶段, 每个阶段要处理的问题都容易被理解和处理。
 - 按自顶向下的方法对问题进行分析、模块化设计和结构化编码等3个步骤。

1. 自顶向下的分析方法

- 把大的复杂的问题分解成小问题后再解决
 - 面对一个复杂的问题，首先进行上层（整体）的分析，按组织或功能将问题分解成子问题
 - 如果子问题仍然十分复杂，再做进一步分解，直到处理对象相对简单，容易处理为止。
 - 当所有的子问题都得到了解决，整个问题也就解决了。
- 每一次分解都是对上一层的问题进行细化和逐步求精，最终形成一种类似树形的层次结构，来描述分析的结果。

学生成绩统计程序的层次结构图



一个模块用（若干个）函数实现

3. 结构化编码主要原则

- 经模块化设计后，每一个模块都可以独立编码。编程时应选用顺序、选择和循环三种控制结构
- 对变量、函数、常量等命名时，要见名知意，有助于对变量含义或函数功能的理解。
- 在程序中增加必要的注释，增加程序的可读性。
- 要有良好的程序视觉组织，利用缩进格式
- 程序要清晰易懂，语句构造要简单直接
- 程序有良好的交互性，输入有提示，输出有说明

5.3 复数运算

5.3.1 程序解析

5.3.2 局部变量和全局变量

5.3.3 变量生命周期和静态局部变量

例5-6 分别输入2个复数的实部与虚部，用函数实现计算2个复数之和与之积。

■ 分析

□ 若2个复数分别为：

$$c1=x1+y1i, c2=x2+y2i,$$

□ 则：

$$c1+c2 = (x1+x2) + (y1+y2)i$$

$$c1*c2 = (x1*x2-y1*y2) + (x1*y2+x2*y1)i$$

运行结果

Enter 1st complex number(real and imaginary):1 1
Enter 2nd complex number(real and imaginary):-2 3
addition of complex is -1.000000+4.000000i
product of complex is -5.000000+1.000000i

```
# include <stdio.h>
double
/* 函数 */
void c
```

```
void complex_add(double real1, double imag1, double real2, double imag2);
int main (void)
```

```
{
```

```
    double imag1, imag2, real1, real2; /* 两个复数的实、虚部变量 */
```

```
    printf ("Enter 1st complex number(real and imaginary): ");
```

```
    scanf ("%lf%lf", &real1, &imag1); /* 输入第一个复数 */
```

```
    printf ("Enter 2nd complex number(real and imaginary): ");
```

```
    scanf ("%lf%lf", &real2, &imag2); /* 输入第二个复数 */
```

```
    complex_add (real1, imag1, real2, imag2); /* 求复数之和 */
```

```
    printf ("addition of complex is %f+%fi\n", result_real, result_imag);
```

```
    complex_prod (real1, imag1, real2, imag2); /* 求复数之积 */
```

```
    printf ("product of complex is %f+%fi\n", result_real, result_imag);
```

```
    return 0;
```

```
}
```

❖ 全局变量：案例代码



```
void complex_add (double real1, double imag1,  
    double real2, double imag2)
```

```
{  
    result_real = real1 + real2;  
    result_imag = imag1 + imag2;  
}
```

```
void complex_prod (double real1, double imag1,  
    double real2, double imag2)
```

```
{  
    result_real = real1 * real2 - imag1 * imag2;  
    result_imag = real1 * imag2 + real2 * imag1;  
}
```

5.3.2 局部变量和全局变量

■ 局部变量

- 在函数内定义的变量（包括形参）

作用范围：本函数内部

- 定义在复合语句内的变量

作用范围：复合语句内部

■ 全局变量

在函数以外定义的变量，不从属于任一函数。

作用范围：从定义处到源文件结束（包括各函数）

例5-6 在复合语句中定义局部变量。

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

```
int main (void)
```

```
{
```

```
    int a;
```

```
    a = 1;
```

```
    {
```

```
        int b = 2;
```

```
        b = a + b;
```

```
        a = a + b;
```

```
    }
```

```
    printf ("%d ", a);
```

```
    return 0;
```

```
}
```

输出:

4

/* 复合语句开始 */

b:小范围内的临时变量

/* 复合语句结束 */

打印b会如何?

❖ 局部变量的作用域

例5-7 全局变量定义

```
#include "stdio.h"
int x;          /* 定义全局变量x */
int f( )
{
    int x = 4;   /* x为局部变量 */
    return x;
}
int main(void)
{
    int a = 1;
    x = a;        /* 对全局变量 x 赋值 */
    a = f( );     /* a的值为4 */
    {
        int b = 2;
        b = a + b; /* b的值为4 */
        x = x + b; /* 全局变量运算 */
    }
    printf("%d %d" , a, x);
    return 0;
}
```

若局部变量与全局变量同名，局部变量优先

输出：
4, 7

❖ 全局变量的作用域

变量作用范围示例

```
int x=1;  
void main( )  
{ int a=2;  
  .....  
  {  
    .....  
    int b=3;  
    .....  
  }  
  f( );  
  .....  
}
```

x=? a=? b=?

b=?

```
int t=4 ;  
void f( )  
{ int x=5, b=6;  
  .....  
}
```

x=5 b=6 t=4 a没定义

x=? b=? t=? a=?

```
int a=7;
```

■ 【例5-8】

- 用函数实现财务现金记账。先输入操作类型(1收入, 2支出, 0结束), 再输入操作金额, 计算现金剩余额, 经多次操作直到输入操作为0结束。要求定义并调用函数, 其中现金收入与现金支出分别用不同函数实现。

■ 分析:

- 设变量**cash**保存现金余额值, 由于它被主函数、现金收入与现金支出函数共用, 任意使用场合其意义与数值都是明确和唯一的, 因此令其为全局变量。

/ 定义计算现金收入函数 */*

```
#include <stdio.h>
double cash;
void income(double number)
{    cash = cash + number;    /* 改变全局变量cash */
}

/* 定义计算现金支出函数 */
void expend(double number)
{    cash = cash - number;    /* 改变全局变量cash */
}

cash = 0;
```

```
printf ("Enter operate choice(0--end, 1--income, 2--expend):");
```

```
scanf ("%d", &choice);    /* 输入操作类型 */
```

```
while (choice != 0){    /* 若输入类型为0，循环结束 */
```

```
    if (choice == 1 || choice == 2) {
```

```
        printf ("Enter cash value:");    /* 输入操作现金额 */
```

```
        scanf ("%d", &value);
        printf ("Enter operate choice(0--end, 1--income, 2--expend):1
```

```
        if (choice == 1) {
            printf ("Enter cash value:1000
```

```
            else {
                printf ("current cash:1000.000000
```

```
            printf ("Enter operate choice(0--end, 1--income, 2--expend):2
```

```
            if (choice == 2) {
                printf ("Enter cash value:456
```

```
                else {
                    printf ("current cash:544.000000
```

```
                scanf ("%d", &value);
                printf ("Enter operate choice(0--end, 1--income, 2--expend):0
```

```
            }
        }
    }
    return 0;
```

```
}
```

❖ 案例代码与运行

5.3.2 局部变量和全局变量

■ 讨论

- 全局变量比局部变量自由度大，更方便？

■ 引起注意

- 对于规模较大的程序，过多使用全局变量会带来副作用，导致各函数间出现相互干扰。如果整个程序是由多人合作开发，各人都按自己的想法使用全局变量，相互的干扰可能会更严重。
- 因此在变量使用中，应尽量使用局部变量，从某个角度看使用似乎受到了限制，但从另一个角度看，它避免了不同函数间的相互干扰，提高了程序质量。
- 局部变量降低函数之间的耦合度间

5.3.3 变量生命周期和静态局部变量

■ 变量生命周期（是否占用存储单元）

变量从定义开始分配存储单元，到运行结束存储单元被回收的整个过程。

■ 自动变量（**auto**）：普通的局部变量

`int x, y;` \longleftrightarrow `auto int x, y;`

`char c1;` \longleftrightarrow `auto char c1;`

□ 函数调用时，定义变量，分配存储单元。

□ 函数调用结束，**收回存储单元**。

■ 全局变量：从程序执行开始，到程序的结束，**存储单元**始终保持。

C程序存储分布示意图（例5-6）

系统存储区

操作系统（如 windows）、语言系统（如 Visual C++）

程序区（C 程序代码）

如主函数、函数 `complex_add()` 、函数 `complex_prod()`等

用户存储区

数据区

静态存储区

全局变量

如 `result_real`, `result_imag`

静态局部变量

动态存储区
（如自动变量）

`main()`变量区:

`real1`, `imag1`, `real2`, `imag2`

`complex_add()`变量区:

`real1`, `imag1`, `real2`, `imag2`

.....

静态局部变量

static 类型名 变量表

- 作用范围：局部变量
- 生命周期：全局变量

```
double fact_s (int n)
```

【例】
义

```
static double f = 1;
```

/* 定义静态变量，第一次赋值为1 */

```
f = f * n;
```

/* 在上一次调用时的值上乘n */

```
return (f);
```

```
}
```

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

```
double fact_s (int n);
```

```
int main (void)
```

```
{
```

```
int i, n;
```

```
printf ("Input n:");
```

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

```
for (i = 1; i <= n; i++)
```

```
printf ("%3d!=%.0f\n", i, fact_s (i));
```

/* 输出i和i! */

```
return 0;
```

```
}
```

fact_s()函数中并没有循环语句，它是靠静态变量f保存着上次函数调用时，计算得到的(n-1)!值，再乘上n，实现n! 的计算。

静态局部变量

- 自动变量如果没有赋初值，其存储单元中将是随机值。
- 就静态变量而言，如果定义时没有赋初值，系统将自动赋0。
- 赋初值只在函数第一次调用时起作用，以后调用都按前一次调用保留的值使用。
- 静态局部变量受变量作用范围限制，不能作用于其他函数（包括主函数）。

静态局部变量

- 静态变量与全局变量均位于静态存储区
 - 他们的共同点是生命周期相同，贯穿整个程序执行过程。
 - 区别在于作用范围不同，全局变量可作用于所有函数，静态变量只能用于所定义函数，而不能用于其他函数。

各类变量生命周期与作用范围示例

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

```
int x=2;
```

```
int f( int x )
```

```
{
```

```
    static int b=2;
```

x=?

```
    b = b+x; x++;
```

```
    return x+b;
```

b=?

```
}
```

```
int main( )
```

```
{ int i, a=1, b=3;
```

x=?

2#9

```
    for( i = 0; i<3; i++ ) {
```

a=?

2#24

```
        b = f(b);
```

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

b=?

2#63

```
    }
```

```
    printf("%d\n", f(a+1) );
```

f()=?

43

```
}
```

❖ 各类变量使用示例

本章小结

■ 系统介绍函数的定义和函数调用

- 学习如何针对具体问题，确定需要使用函数的功能要求，再将功能用函数程序实现
- 考虑如何调用定义好的函数，实现主调函数与被调函数的连接
- 确定参数功能，掌握参数的传递实现

■ 函数与变量间的关系，不同形式的变量在函数中起的作用不同。

- 局部变量、全局变量和静态变量



Chap 6 (回顾)数据类型和表达式

6.1 数据的存储和基本数据类型

6.2 数据的输入和输出

6.3 类型转换

6.4 运算符与表达式

本章要点

- 整数类型的补码表示
- 整数类型的其它变型以及不同进制的常量表示
- 字符型变量、常量（含转义符）与**ASCII**码的关系及其运算
- 浮点数系、精度与取值范围、实型常量
- 三种基本类型数据的输入和输出
- 类型转换：自动（赋值、非赋值）、强制
- 算术运算、赋值运算、关系运算、逻辑运算、条件运算和逗号运算
- 各类运算的优先级与结合性、运算及表达式的特点
- 位运算（不作要求）

数据类型和表达式

■ 数据类型

□ 基本数据类型

- 整型 (**int**、**unsigned**、**signed**、**short**、**long**)
- 字符型 (**char**)
- 实型 (浮点型) (**float**、**double**)

□ 构造数据类型

数组、结构 (**struct**)、联合 (**union**, 略)、枚举 (**enum**, 略)

□ 指针类型 (*)

□ 空类型 (**void**)

■ 运算：对数据的操作

运算符 + 数据 → 表达式

6.1 数据的存储和基本数据类型

6.1.1 数据的存储

整型、字符型、实型数据的存储

6.1.2 基本数据类型

整型与整型常量（整数）

字符型与字符型常量

实型与实型常量（实数）

6.1.1 数据的存储—整型数据

设整数在内存中用2个字节存储

1 000 0001 1000 0001

0 000 0001 1000 0001

符号位

1: 负数

0: 正数

数值的表示方法—原码 反码 补码

■ 正数的原码、反码和补码相同

1 的补码 0 000 0000 0000 0001

.....

32767 的补码 0 111 1111 1111 1111

($2^{15}-1$, 2个字节的存储单元能表示的最大正数)

■ 负数的原码、反码和补码不同

-1

□ 原码 1 000 0000 0000 0001

□ 反码 1 111 1111 1111 1110 原码取反（符号位保持不变）

□ 补码 1 111 1111 1111 1111 反码+1

原码 反码 补码

32767

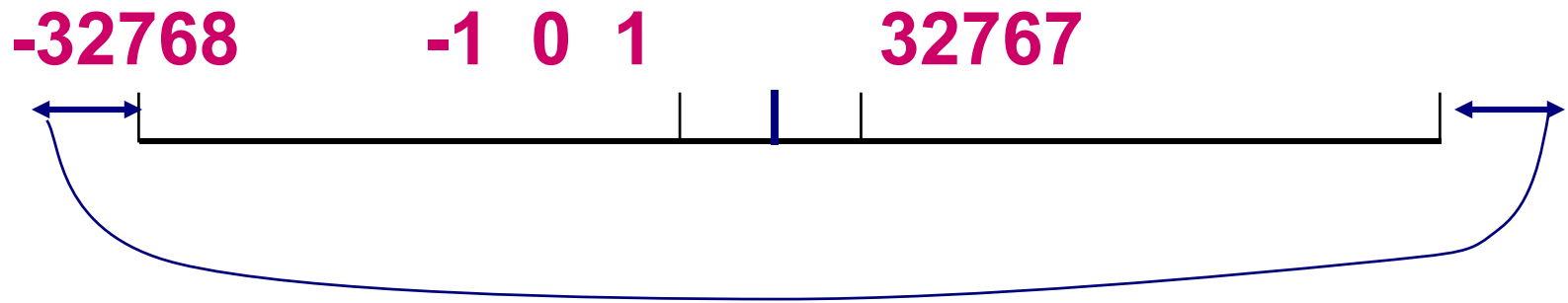
- 补码 **0** 111 1111 1111 1111

-32767

- 原码 **1** 111 1111 1111 1111
- 反码 **1** 000 0000 0000 0000 原码取反（符号位保持不变）
- 补码 **1** 000 0000 0000 000**1** 反码+1

-32768 = -32767-1

- 补码 **1** 000 0000 0000 0000
(2个字节的存储单元能表示的最小负数)



$$32767 + 1 = 32768 ?$$

$$1000\ 0000\ 0000\ 000 = -32768$$

32767 0111 1111 1111 1111

.....

1 0000 0000 0000 0001

0 0000 0000 0000 0000

-1 1111 1111 1111 1111

-2 1111 1111 1111 1110

.....

-32767 1000 0000 0000 0001

-32768 1000 0000 0000 0000

$$-32768 - 1 = -32769 ?$$

$$0111\ 1111\ 1111\ 1111 = 32767$$

6.1.2 基本数据类型

■ 整型

有符号整型	无符号整型	数据长度
<code>int</code>	<code>unsigned [int]</code>	16或32位
<code>short [int]</code>	<code>unsigned short [int]</code>	16位
<code>long [int]</code>	<code>unsigned long [int]</code>	32位

■ 字符型

`char` 8位（1个字节）

■ 实型（浮点型）

单精度浮点型	<code>float</code>	32位
双精度浮点型	<code>double</code>	64位

基本数据类型—整型

扩展的整数类型: **short long unsigned [int]**

有符号整型	无符号整型	数据长度
int	unsigned [int]	16或32位
short [int]	unsigned short [int]	16位
long [int]	unsigned long [int]	32位

有符号 **short**

1 000 0000 0000 0000 **-32768** **-2^{15}**

0 111 1111 1111 1111 **32767** **$2^{15}-1$**

无符号 **unsigned short**

0000 0000 0000 0000 **0**

1111 1111 1111 1111 **65535** **$2^{16}-1$**

❖ 整数类型的其它形式

整数类型的取值范围

int 32位 -2147483648 ~ 2147483647 $-2^{31} \sim 2^{31}-1$
short [int] 16位 -32768 ~ 32767 $-2^{15} \sim 2^{15}-1$
long [int] 32位 -2147483648 ~ 2147483647 $-2^{31} \sim 2^{31}-1$
[signed] 表示有符号，亦可省略

unsigned [int] 32位 0 ~ 4294967295 $0 \sim 2^{32}-1$
unsigned short [int] 16位 0 ~ 65535 $0 \sim 2^{16}-1$
unsigned long [int] 32位 0 ~ 4294967295 $0 \sim 2^{32}-1$

整型常量（整数）

■ 整数的表示

三种表现形式(合法符号):

- 十进制整数：正、负号，**0~9**，首位**不是0**
例：**10, 123**
- 八进制整数：正、负号，**0~7**，首位是**0**
例：**010, 0123**
- 十六进制整数：正、负号，**0~9, a~f, A~F**，前缀是**0x, 0X**
例：**0x10, 0X123**

整数的表示

123 = 01111011 (B) 二进制
=173 (O) 八进制
=7B (X) 十六进制

123 0173 0x7b

16 020 0X10

10 012 0XA

10 010 0x10

- 不能超出整型数据的取值范围
- 比长整型数还要大的数只能用实数来表示

整数的类型

判断整数的类型

- 整数后的字母后缀

- 123L long

- 123U unsigned

- 123LU unsigned long

- 整数的值

基本数据类型—字符型

■ 字符具有数值特征

'A' 65 0100 0001 (1个字节存储)

■ 整型变量和字符变量的定义和赋值可以互换 【ASCII码范围】

```
char c;  
c = 'A';  
或  
c = 65;
```

```
int i;  
i = 65;  
或  
i = 'A';
```


字符型常量

■ 字符常量

'a'、'A'、'9'、'+'、'\$'（注意：单引号）

■ ASCII字符集

列出所有可用的字符（**256**个）

- 每个字符：惟一的次序值（**ASCII** 码）
- '0' ~ '9' 升序排列
- 'A' ~ 'Z'
- 'a' ~ 'z'

字符的数值特征

字符—ASCII 码

对字符进行运算：对字符的**ASCII** 码进行运算

例如：

区分数字符和数字

'1'

1

'A' 的 ASCII 码 65

则：**'A'+1=66**，对应字符 **'B'**

转义字符

字符形式	所表示字符
\n	换行
\t	横向跳格（即输出若干个空格）
\b	退格（显示输出时，刷新左边一个字符）
\r	回车（输出位置重新移到行首）
\\	反斜杠字符“\”
\'	单引号（撇号）
\ddd	八进制数 ddd 所代表字符，如\007 为“嘟”声，\40 即空格。
\xhh	十六进制数 hh 所代表的字符，如\x41 即'A'，\x20 即空格。

- 反斜杠后跟一个字符或数字
- 字符常量，代表一个字符

‘\n’ ‘A’ ‘\101’ ‘\x41’

\最多3位8进制数字，值不超过十进制255

- 所有字符都可以用转义字符表示

❖ 转义字符\\

基本数据类型—实型

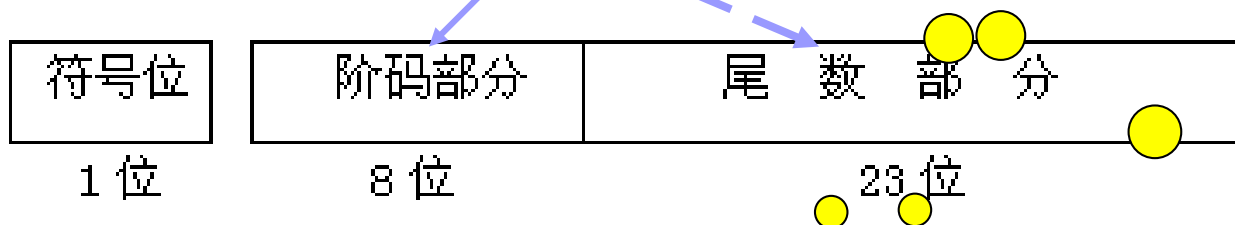
实型（浮点型）数据

- 单精度浮点型 **float**
- 双精度浮点型 **double**

	存储 (字节数)	数据精度 (有效数字)	取值范围 (最大最小数)
float	4字节	约7/8位	$\pm(10^{-38} \sim 10^{38})$
double	8字节	约16位	$\pm(10^{308} \sim 10^{308})$

浮点数系

$$x = 0.t_1t_2\dots t_n \times 10^{\pm e}$$



👉 尾数部分的位数 n （即有效位数, $t_1 \neq 0$ ）决定了精度

👉 阶码部分 e 决定了取值范围（ $e=38/308$: float/double）

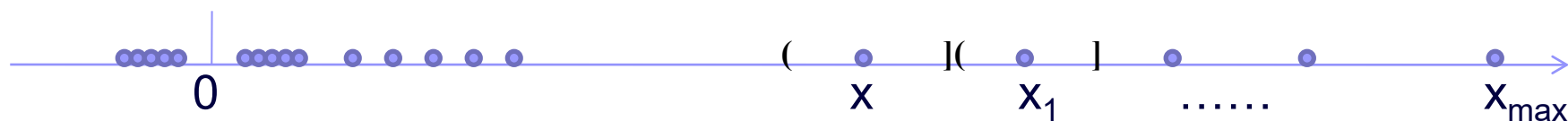
如: $X_{\max} = \pm 0.9999999 \times 10^{+38}$, $X_{\min} = \pm 0.5000000 \times 10^{-38}$

👉 浮点数表示实数只是近似表示——计算会产生误差并传播

👉 一方面可以增加字长提高精度, 另一方面要研究误差传播规律

用浮点数表示实数

浮点数是有限而稀疏的：



假设浮点数 $x = 123456.70 = 0.1234567 \times 10^6$,

比它大的下一个浮点数是： $x_1 = 0.1234568 \times 10^6$
即 $x_1 = 123456.80$ ，所以

区间 $(x-0.05, x+0.05]$ 中的所有实数用 x 近似表示，

区间 $(x_1-0.05, x_1+0.05]$ 中的所有实数用 x_1 近似表示。

👉 超过 x_{\max} 太多的实数将会溢出。

数据精度和取值范围

- **数据精度** 与 **取值范围** 是两个不同的概念：

float x = 1234567.89;

1234567.80

虽在取值范围内，但无法精确表达。

float y = 1.2e55;

y 的精度要求不高，但超出取值范围。

- 并不是所有的实数都能在计算机中精确表示
- 实型常量的类型都是 **double**

实型常量（实数、浮点数）

■ 实数的表示

□ 浮点表示法

0.123 123.4 12. .12

□ 科学计数法

6.026E-27 1.2e+30 1E-5

6.2 数据的输入和输出

6.2.1 整型数据的输入和输出

6.2.2 实型数据的输入和输出

6.2.3 字符型数据的输入和输出

6.2.1 整型数据的输入输出

printf (格式控制, 输出参数1, ..., 输出参数n);

scanf (格式控制, 输入参数1, ..., 输入参数n);

格式控制说明 %...

	十进制	八进制	十六进制
int	%d	%o	%x
long	%ld	%lo	%lx
unsigned	%u	%o	%x
unsigned long	%lu	%lo	%lx

输出整型数据示例（1）

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

```
int main(void)
```

```
{
```

```
    printf("%d, %o, %x\n", 10, 10, 10);
```

```
    printf("%d, %d, %d\n", 10, 010, 0x10);
```

```
    printf("%d, %x\n", 012, 012);
```

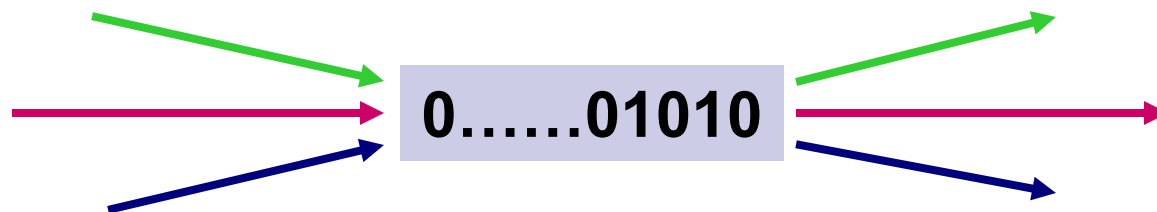
```
    return 0;
```

```
}
```

10, 12, a

10, 8, 16

10, a



输入整型数据示例（2）

```
# include <stdio.h>
int main(void)
{
    int a, b;

    printf("input a, b:");
    scanf( "%x%d" , &a, &b);
    printf("%d%5d\n", a, b);
    printf("%x, %d\n", a, b);
    return 0;
}
```

input a, b: 17 17
15 17
f, 17

/*%5d指定变量b的输出宽度为5 */

6.2.2 实型数据的输入和输出

■ 输入 `scanf()`

- **float**: `%f` 或 `%e`

以小数或指数形式输入一个单精度浮点数

- **double**: `%lf` 或 `%le`

以小数或指数形式输入一个双精度浮点数

■ 输出 `printf()`

float 和 **double** 使用相同的格式控制说明

- `%f`

以小数形式输出浮点数，保留6位小数

- `%e`

以指数形式输出

实型数据输出格式示例

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

```
int main(void)
```

```
{
```

```
    double d = 3.1415926;
```

```
    printf("%f, %e\n", d, d);
```

```
    printf("%5.3f, %5.2f, %.2f\n", d, d, d);
```

```
    return 0;
```

```
}
```

3.141593, 3.14159e+00

3.142, 3.14, 3.14

↑
一共5位，小数部分3位，小数点占1位

实型数据输入输出示例

假定float的精度为7位，double的精度为16位

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

```
int main(void)
```

```
{
```

```
    float f;
```

```
    double d;
```

```
    printf("input f, d:");
```

```
    scanf("%f%lf", &f, &d);
```

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

```
    d = 1234567890123.12;
```

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

```
    return 0;
```

```
}
```

input f, d:

1234567890123.123456

1234567890123.123456

f = 1234567954432.000000

d = 1234567890123.123540

d = 1234567890123.120120

6.2.3 字符型数据输入输出

■ scanf() 和 printf()

%c

char ch;

scanf("%c", &ch);

printf("%c", ch);

■ getchar() 和 putchar()

char ch;

ch = getchar();

putchar(ch);

输入输出一个字符

输入输出字符示例

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

```
int main(void)
```

```
{
```

```
    char ch1, ch2;
```

```
    ch1=getchar();
```

```
    ch2=getchar();
```

```
    putchar(ch1);
```

```
    putchar('#');
```

```
    putchar(ch2);
```

```
    return 0;
```

```
}
```

Ab

A#b

输入输出字符示例

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

```
int main(void)
```

```
{
```

```
    char ch1, ch2, ch3;
```

```
    scanf("%c%c%c", &ch1, &ch2, &ch3);
```

```
    printf("%c%c%c%c%c", ch1, '#', ch2, '#', ch3);
```

```
    return 0;
```

```
}
```

AbC

A#b#C

A bC

A# #b

【例6-1】 单词加密解析。输入一个英文单词（由6个小写英文字母组成），按照下列过程将该单词加密：

先将英文单词中的小写字母转换为对应的大写字母，再将该大写字母的ASCII码对10整除后取其余数，从而得到一个六位整数密码（不可逆）。

```
# include <stdio.h>
int main (void)
{   int i;
    char ch_lower, ch_upper;
    for (i = 1; i <= 6; i++) {
        scanf ("%c", &ch_lower);
        if (ch_lower >= 'a' && ch_lower <= 'z')
            ch_upper = ch_lower - 'a' + 'A';
        /*将小写字母转换为大写字母*/
        printf ("%c->%c->%d\n", ch_lower, ch_upper, ch_upper%10);
    }
    return 0;
}
```

friday
f->F->0
r->R->2
i->I->3
d->D->8
a->A->5
y->Y->9

字符运算

■ 大小写英文字母转换

$'b' - 'a' = 'B' - 'A'$

.....

$'z' - 'a' = 'Z' - 'A'$

$'m' \leftrightarrow 'M'$

$'a' \rightarrow 'A'$
 $'A' \rightarrow 'a'$

● $'m' - 'a' + 'A' = 'M'$

● $'M' - 'A' + 'a' = 'm'$

■ 数字字符和数字转换

$9 - 0 = '9' - '0'$

$'9' = 9 + '0'$

$'8' \leftrightarrow 8$

$'8' \rightarrow 8$
 $8 \rightarrow '8'$

● $'8' - '0' = 8$

● $8 + '0' = '8'$

6.3 类型转换

不同类型数据的混合运算，先转换为同一类型，再运算。

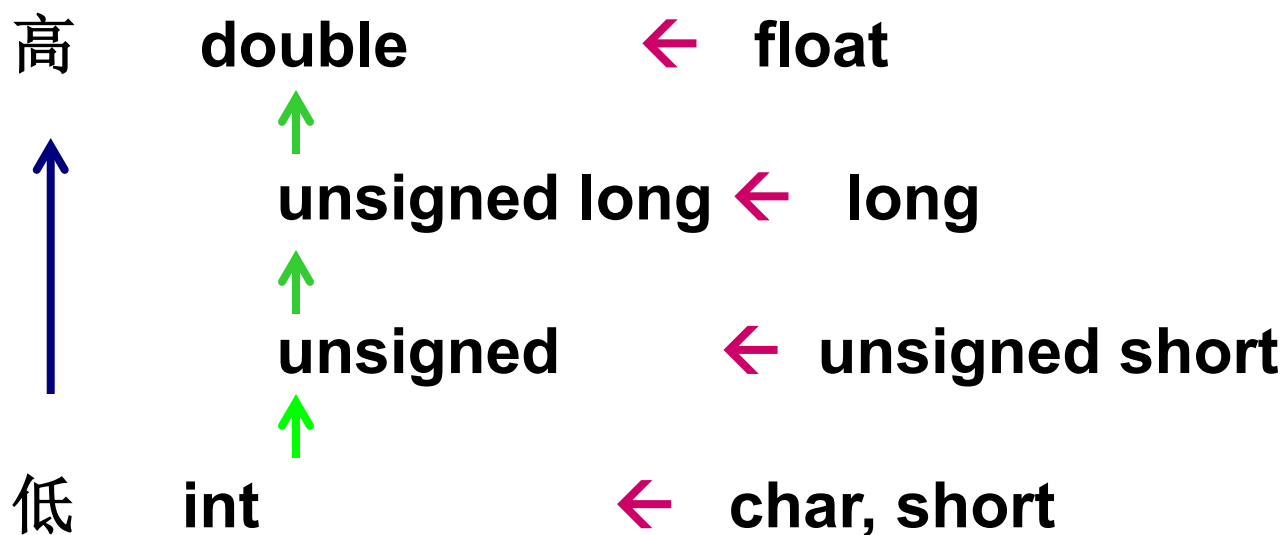
6.3.1 自动类型转换

- 非赋值运算的类型转换
- 赋值运算的类型转换

6.3.2 强制类型转换

6.3.1 自动类型转换（非赋值运算）

- 水平方向：自动
- 垂直方向：低 → 高



自动类型转换（非赋值运算）

'A' + 12 - 10.05

65

77

66.95

高



低

double



unsigned long



unsigned



int

← float

← long

← unsigned short

← char, short

自动类型转换（赋值运算）

变量 = 表达式

- 计算赋值运算符右侧**表达式**的值
- 将赋值运算符右侧**表达式**的值赋给左侧的**变量**

将赋值运算符右侧表达式的类型
自动转换成
赋值号左侧变量的类型

自动类型转换（赋值运算）

```
double x;
```

x = 1; **x = ?**

```
short a = 1000;
```

```
char b = 'A';
```

```
long c;
```

c = a + b; **c = ?**

```
int ai;
```

ai = 2.56; ai = ?

```
short bi;
```

bi = 0x12345678L

bi = ?

6.3.2 强制类型转换

强制类型转换运算符
(类型名) 表达式

(double)3

3.0

(int)3.8

3

(double)(5/2)

2.0

(double)5/2

2.5

强制类型转换示例

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

```
int main (void)
```

```
{
```

```
    int i;
```

```
    double x;
```

```
    x = 3.8;
```

```
    i = (int) x;
```

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

```
    printf ("(double)(int)x = %f\n", (double)(int)x);
```

```
    printf (" x mod 3 = %d\n", (int)x % 3);
```

```
    return 0;
```

```
}
```

$x = 3.800000, i = 3$

$(\text{double})(\text{int})x = 3.000000$

$x \bmod 3 = 0$

四舍五入？

$i = (\text{int}) (x + 0.5);$

6.4 表达式

表达式：由运算符和运算对象（操作数）组成的有意义的运算式子，它的值和类型由参加运算的运算符和运算对象决定。

- 运算符：具有运算功能的符号
- 运算对象：常量、变量和函数等表达式

算术表达式、赋值表达式、关系表达式、逻辑表达式、条件表达式和逗号表达式等

6.4.1 算术表达式—算术运算符

■ 单目 **+** **-** **++** **--**

■ 双目 **+** **-** ***** **/** **%**

注意

□ **/** 整数除整数，得整数

$$1/4 = 0, 10/3 = 3$$

□ **%** 模(求余)： 针对整型数据

$$5\%6 = 5, 9\%4 = 1, 100\%4 = 0$$

□ **+** 和 **-**

■ 单目运算符， **+10** 和 **-10**

■ 双目运算符， **x+10** 和 **y-10**

□ 双目运算符两侧操作数的类型要相同，否则，自动类型转换后，再运算。

自增运算符++和自减运算符--

`int n;`

`n++ ++n n-- --n`（只适合变量运算）

□ 使变量的值增1或减1

`++n n++` `n = n + 1`

`--n n--` `n = n - 1`

□ 取变量的值作为表达式的值

`++n`: `n = n + 1`; 取`n`值作为表达式 `++n` 的值

`n++`: 取`n`值作为表达式 `n++` 的值; `n = n + 1`

自增运算和自减运算

```
int n, m;
```

```
n=n+1  
m=n
```

```
n=2;  
m=++n;
```

```
n=3
```

```
m=3
```

```
m=n  
m=n+1
```

```
n=2;  
m=n++;
```

```
n=3
```

```
m=2
```

算术运算符的优先级和结合性

单目 $+$ $-$ $++$ $--$

高

从右向左

双目 $*$ $/$ $\%$



双目 $+$ $-$

低

$$-5 + 3\%2 = (-5) + (3\%2) = -4$$

$$3 * 5 \% 3 = (3*5) \% 3 = 0$$

$-i++$

$-(i++)$

写出C表达式

数学式 \rightarrow C算术表达式

$$s(s-a)(s-b)(s-c)$$

$$(x+2)e^{2x}$$

$$\frac{-b + \sqrt{b^2 - 4ac}}{2a}$$

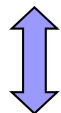
6.4.2 赋值表达式

■ 赋值运算符 =

$x = 3 * 4$

优先级较低，结合性从右向左

$x = y = 3$



$x = (y = 3)$

赋值表达式

右侧表达式的类型自动转换成左侧变量的类型

变量 = 表达式

- 计算赋值运算符右侧**表达式**的值
- 将赋值运算符右侧**表达式**的值赋给左侧的**变量**
- 将赋值运算符左侧的**变量**的值作为表达式的值

int n;

double x, y;

n = 3.14 * 2;

x = 10 / 4;

x = (y = 3);

复合赋值运算符

■ 赋值运算符

- 简单赋值运算符 **=**

- 复合赋值运算符

 - 复合算术赋值运算符 **+= -= *= /= %=**

 - 复合位赋值运算符

■ 赋值表达式

变量 **赋值运算符** 表达式

x += exp 等价于 **x = x + exp**

x *= y - 3 **x = x * (y-3)**

6.4.3 关系表达式—关系运算符

- 比较两个操作数，比较的结果：**真 假**

$x < y$ $x \leq y$ $x == y$

$x > y$ $x \geq y$ $x != y$

- 优先级

- ☐ 算术运算符

- ☐ $< \leq > \geq$

- ☐ $== !=$

- ☐ 赋值运算符

- 左结合

$a > b == c$

$d = a > b$

$ch > 'a' + 1$

$d = a + b > c$

$3 \leq x \leq 5$

$b - 1 == a != c$

$(a > b) == c$

$d = (a > b)$

$ch > ('a' + 1)$

$d = ((a + b) > c)$

$(3 \leq x) \leq 5$

$((b - 1) == a) != c$

关系表达式

- 用关系运算符将2个表达式连接起来的式子
哪些是关系表达式？

`a > b == c` 0

`d = a > b` 0

`ch > 'a' + 1` 1

`d = a + b > c` 1

`b - 1 == a != c` 0

`3 <= x <= 5` 1

`char ch = 'w';`

`int a = 2, b = 3, c = 1, d, x=10;`

- 关系运算的结果

☐ 真 1

☐ 假 0

6.4.4 逻辑表达式—逻辑运算符

`(ch >= 'a' && ch <= 'z') || (ch >= 'A' && ch <= 'Z')`

`ch == ' ' || ch == '\n'`

`x >= 3 && x <= 5`

■ && || !

■ 逻辑运算结果: **1**(真) **0** (假)

■ 逻辑运算对象: 关系表达式或逻辑量

`x >= 3 && x <= 5` `!x`

■ 判断逻辑量的真假: **非0** (真) **0** (假)

逻辑运算的规则—真值表

x	y	x&&y	x y	!x
0	0	0	0	1
0	非0	0	1	1
非0	0	0	1	0
非0	非0	1	1	0

逻辑运算符的优先级和结合性

■ 优先级

□ !

□ 算术运算符

□ 关系运算符

□ &&

□ ||

□ 赋值运算符

a || b && c

!a && b

x >= 3 && x <= 5

!x == 2

a || 3 + 10 && 2

a || (b && c)

(!a) && b

(x >= 3) && (x <= 5)

(!x) == 2

a || ((3 + 10) && 2)

■ 左结合

逻辑表达式

用**逻辑运算符**将**关系表达式**或**逻辑量**连接起来的式子

哪些是逻辑表达式？

a && b 0

a || b && c 1

!a && b 0

a || 3+10 && 2 1

!(x == 2) 1

!x == 2 0

ch || b 1

```
char ch = 'w';
```

```
int a = 2, b = 0, c = 0;
```

```
float x = 3.0;
```

exp1 && exp2

先算 **exp1**，若其值为 0，
STOP

exp1 || exp2

先算 **exp1**，若其值为 1，
STOP

例[6-4]写出满足要求的逻辑表达式

■ x 为零

□ 关系表达式 $x == 0$
等价

□ 逻辑表达式 $!x$

{ x 取0 $x==0$ 真
 x 取非0 $x==0$ 假

{ x 取0 $!x$ 真
 x 取非0 $!x$ 假

■ x 不为零

□ $x != 0$

□ x

■ x 和 y 不同时为零

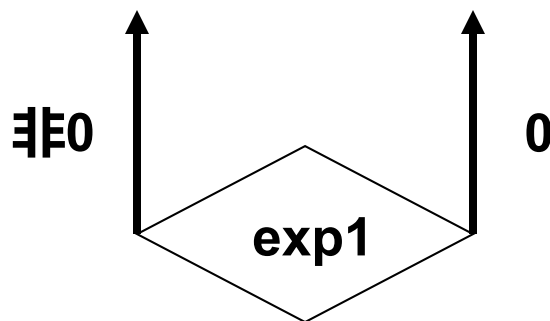
□ $!(x == 0 \ \&\& \ y == 0)$

□ $x != 0 \ || \ y != 0$

□ $x \ || \ y$

6.4.5 条件表达式

exp1 ? exp2 : exp3



$$y = \begin{cases} x+2 & x>0 \\ x^2 & x\leq 0 \end{cases}$$

$y = (x>0) ? x+2 : x*x;$

int n;

(n>0) ? 2.9 : 1

n = 10 2.9

n = -10 1.0

if (a>b)

z = a;

else

z = b;

if (x>0)

y=x+2;

else

y=x*x;

$z = (a>b) ? a : b;$

6.4.6 逗号表达式

表达式1, 表达式2,, 表达式n

先计算表达式 1 , 然后计算表达式 2 ,, 最后计算表达式n的值, 并将表达式n的值作为逗号表达式的值.

```
int a, b, c, x;
```

```
(a=2), (b=3), (c=a+b);
```

```
x=(a=2, b=3, c=a+b);
```

```
a=2;
```

```
printf("a=%d,b=%d", a, (b=3, c=a*b));
```

逗号运算符的优先级最低, 左结合

逗号表达式的用途

```
sum = 0;  
for (i = 0; i <= 100; i++)  
    sum = sum + i;
```

```
for (i = 0, sum = 0; i <= 100; i++)  
    sum = sum + i;
```

```
for (i = 0, sum = 0; i <= 100; i++)  
    sum += i;
```

6.4.8 其他运算

■ 长度运算符 **sizeof**

单目运算符，计算变量或数据类型的字节长度

int a;

sizeof(a)

求整型变量 a 的长度，值为4(bytes)

sizeof(int)

求整型的长度，值为4 (bytes)

sizeof(double)

求双精度浮点型的长度，值为8 (bytes)

运算符的优先级和结合性

■ **() []**

← ■ **! - + ++ -- (类型名) sizeof**

■ *** / %**

■ **+ -**

■ **< <= > >=**

■ **== !=**

■ **&&**

■ **||**

← ■ **? :**

← ■ **= += -= *= /= %=**

■ **,**

6.4.9 程序解析—大小写字母转换

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

```
int main(void)
```

```
{
```

```
    char ch;
```

```
    printf("input characters: ");
```

```
    ch = getchar();
```

```
    while(ch != '\n'){
```

```
        if(ch >= 'A' && ch <= 'Z') ch = ch - 'A' + 'a';
```

```
        else if((ch >= 'a' && ch <= 'z' ) ch = ch - 'a' + 'A';
```

```
        putchar(ch);
```

```
    ch = getchar();
```

```
}
```

```
    return 0;
```

```
}
```

input 10 characters: **Reold 123?**

rEOLD 123?

`while((ch = getchar()) != '\n')`

`(ch = getchar()) != '\n'`

`ch = getchar() != '\n'`

等价吗?

6.4.7 位运算（不作要求）

■ 位逻辑运算

~ 按位取反 单目 右结合

& 按位与

^ 按位异或：相同取0，不同取1

| 按位或

■ 移位运算

<< 对操作数左移给出的位数

>> 对操作数右移给出的位数

■ 复合位赋值运算

位逻辑运算

~ 按位取反

& 按位与

^ 按位异或：相同取0，不同取1

| 按位或

x=0 00000000 00000000

y=3 00000000 00000011

x & y 00000000 00000000

x | y 00000000 00000011

x ^ y 00000000 00000011

1010 ^ 0101 = 1111

注意区分：

& 和 |

&& 和 ||

x && y 得 0

x || y 得 1

位移位运算

<< 对操作数左移给出的位数

>> 对操作数右移给出的位数

x<<3 将x向左移3位，空出的位用零填补

00111010 << 3

11010000

x>>3 将x向右移3位

00111010 >> 3

00000111

复合位赋值运算符

&=

|=

^=

>>=

<<=

a &= b 相当于 a = a & b

a <<= 2 相当于 a = a << 2

Chap 7 数 组

7.1 一维数组：输出所有大于平均值的数

7.2 二维数组：找出矩阵中最大值所在的位置

7.3 字符串：判断回文

本章要点

- 什么是数组？为什么要使用数组？如何定义数组？
- 如何引用数组元素？
- 二维数组的元素在内存中按什么方式存放？
- 什么是字符串？字符串结束符的作用是什么？
- 如何实现字符串的存储和操作，包括字符串的输入和输出？
- 怎样理解C语言将字符串作为一个特殊的一维字符数组？

7.1 输出所有大于平均值的数

例7-1 输入10个整数，计算这些数的平均值，再输出所有大于平均值的数。

7.1.1 程序解析

7.1.2 一维数组的定义和引用

7.1.3 一维数组的初始化

7.1.4 使用一维数组编程

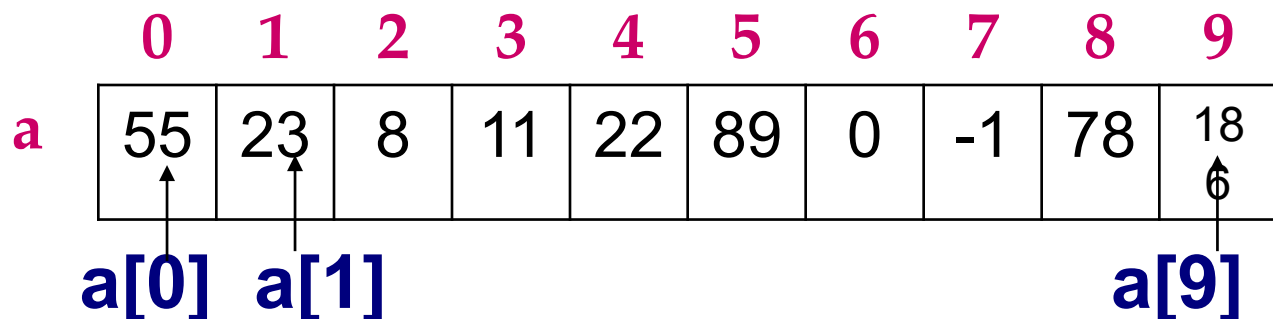
7.1.1 程序解析—输出大于均值的数

```
int main(void)
{ int i; double average, sum;  /* average存放平均值 */
  int a[10];                  /* 定义1个数组a，它有10个整型元素 */
  printf ("Enter 10 integers: ");
  sum = 0;
  for( i = 0; i < 10; i++ ){
    scanf ( "%d", &a[i] );
    sum = sum + a[i];
  }
  average = sum / 10;
  printf ( "average = %.2f\n", average );
  printf ( ">average:" );
  for( i = 0; i < 10; i++ ){
    if( a[i] > average )
      printf( "%d ", a[i] );
  }
  printf ( "\n" );
  return 0;
}
```

Enter 10 integers: 55 23 8 11 22 89 0 -1 78 186
average = 47.10
>average: 55 89 78 186

数组

```
for( i = 0; i < 10; i++ ){  
    scanf ( "%d", &a[i] );  
    sum = sum + a[i]; }
```



- 数组：相同类型数据的有序集合，在内存中连续存放。
 - 由数组名和下标唯一地确定每个数组元素
 - 每个元素都属于同一类型
- 一批相同类型的变量使用同一个数组变量名，用下标来相互区分。
 - 优点：表述简洁，可读性高；便于使用循环结构

7.1.2 一维数组的定义和引用

1. 定义

类型名 数组名 [数组长度]

数组长度为常量

类型名：数组元素的类型

数组名：数组（变量）的名称，标识符

数组长度：常量表达式，给定数组的大小

int a[10];

定义一个含有10个整型元素的数组 a

char c[200];

定义一个含有200个字符元素的数组 c

float f[5];

定义一个含有5个浮点型元素的数组 f

2. 数组的内存结构

int a[10];

假设系统规定 **int** 类型占用**2**个字节，则对于数组**a**，其内存分配形式

只要知道了数组第一个元素的地址以及每个元素所需的字节数，其余各个元素的存储地址均可计算得到。

内存地址	下标	值
4028	9	
4026	8	
4024	7	
4022	6	
4020	5	
4018	4	
4016	3	
4014	2	
4012	1	
a 4010	0	

数组名是一个地址常量，存放数组内存空间的首地址。

3. 引用

- 先定义，后使用
- 只能引用单个的数组元素，不能一次引用整个数组

int a[10];

10个数组元素：a[0]、a[1]、..... a[9]

数组元素：数组名[下标]

下标：整型表达式

下标取值范围：**[0，数组长度-1]**

**下标不要越界
不能使用a[10]**

- 数组元素的使用方法与同类型的变量相同

scanf("%d", &a[i]);

sum = sum + a[i];

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

区分数组的定义和数组元素的引用

定义数组

类型名 数组名[数组长度]

引用数组元素

数组名[下标]

```
int a[10];
```

```
a[0] = a[9] = 0;
```

```
a[i] = i;
```

数组长度为常量

下标不要越界

7.1.3 一维数组的初始化

- 定义数组时，对数组元素赋初值

类型名 数组名[数组长度] = {初值表};

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

a[0]=1, a[1]=2,..... a[9]=10

- 静态数组、动态数组的初始化

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

静态存储的数组如果没有初始化，所有元素自动赋0

```
static int b[5];
```

动态存储的数组如果没有初始化，所有元素为随机值

```
auto int c[5]; 等价于 int c[5];
```

针对部分元素的初始化

```
static int b[5] = {1, 2, 3};
```

b[0] = 1, b[1] = 2, b[2] = 3, b[3] = 0, b[4] = 0

```
auto int fib[20] = {0, 1};
```

fib[0] = 0, fib[1] = 1, 其余元素不确定

- 如果对全部元素都赋初值，可以省略数组长度

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

↑
建议不要省略数组长度

7.1.4 使用一维数组编程

数组和循环

```
for ( i = 0; i < n; i++ )  
    printf ( "%d ", a[i] );
```

数组下标作为循环变量，通过循环，逐个处理数组元素

一维数组示例

例 7-2 用数组计算fibonacci数列的前10个数，并按每行打印5个数的格式输出。

例 7-3 顺序查找法。输入5个整数，将它们存入数组a中，再输入1个数x，然后在数组中查找x，如果找到，输出相应的最小下标，否则，输出“Not Found”。

例 7-4 输入n($n < 10$)，再输入n个数

(1) 输出最小值和它所对应的下标

(2) 将最小值与第一个数交换，输出交换后的n个数

例 7-5 选择排序法。输入一个n($1 < n \leq 10$)，再输入n个整数，用选择法将它们从小到大排序后输出。

例 7-6 调查电视节目欢迎程度。某电视台要进行一次对该台8个栏目（设相应栏目编号为1~8）的受欢迎情况，共调查了1000位观众，现要求编写程序，输每一位观众的投票，每位观众只能选择一个最喜欢的栏目投票，统计输出~各栏目的得票情况。

补充 二分查找法。设已有一个10个元素的整形数组a，且按值从小到大有序。输入一个整数x，然后在数组中查找x，如果找到，输出相应的下标，否则，输出“Not Found”。

例 7-2 计算fibonacci数列

用数组计算**fibonacci**数列的前**10**个数，并按每行打印**5**个数的格式输出。

1, 1, 2, 3, 5, 8, 13,

用数组计算并存放**fibonacci**数列的前**10**个数

$$f[0] = f[1] = 1$$

$$f[i] = f[i-1] + f[i-2] \quad 2 \leq i \leq 9$$

例 7-2 源程序

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

```
int main(void)
```

```
{
```

```
    int i;
```

```
    int fib[10] = {1, 1}; /* 数组初始化 */
```

```
    for ( i = 2; i < 10; i++ )
```

```
        fib[i] = fib[i - 1] + fib[i - 2];
```

```
    for ( i = 0; i < 10; i++ ){
```

```
        printf ( "%6d", fib[i] );
```

```
        if ( (i + 1) % 5 == 0 ) /* 5个数换行 */
```

```
            printf("\n");
```

```
    }
```

```
    return 0;
```

```
}
```

1	1	2	3	5
8	13	21	34	55

例7-3 在数组中查找一个给定的数

输入5个整数，将它们存入数组a中，再输入1个数x，然后在数组中查找x，如果找到，输出相应的最小下标，否则，输出“**Not Found**”。

输入：2 9 8 1 9

9

输出：1

输入：2 9 8 1 6

7

输出：Not Found

```

#include <stdio.h>
int main(void)
{
    int i, flag, x;
    int a[5];
    printf ( "Enter 5 integers: " );
    for ( i = 0; i < 5; i++ )
        scanf ( "%d", &a[i] );
    printf ("Enter x: " );
    scanf ( "%d", &x );
    flag = 0;
    for ( i = 0; i < 5; i++ )
        if ( a[i] == x ){
            printf ( "Index is %d\n", i );
            flag = 1;
            break;
        }
    if ( flag == 0 )    printf("Not Found\n");
    return 0;
}

```

例 7-3 源程序

Enter 5 integers: **2 9 8 1 9**
 Enter x: **9**
 Index is 1

Enter 5 integers: **2 9 8 1 9**
 Enter x: **7**
 Not Found



❖ 实例：查找（第一个）

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

```
int main(void)
```

```
{ int i, flag, x; int a[5];
```

```
printf("Enter 5 integers: ");
```

```
for(i = 0; i < 5; i++)
```

```
scanf("%d", &a[i]);
```

```
printf("Enter x: ");
```

```
scanf("%d", &x);
```

```
flag = 0;
```

```
for(i = 0; i < 5; i++)
```

```
if(a[i] == x){
```

```
printf("Index is %d\n", i);
```

```
flag = 1;
```

```
break;
```

```
}
```

```
if(flag == 0) printf("Not Found\n");
```

```
return 0;
```

```
}
```

例 7-3 思考(1)

去掉**break**语句，结果？

Enter 5 integers: 2 9 8 1 9

Enter x: 9

Index is 1

Index is 4

❖ 实例：查找（第几个？）

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

```
int main(void)
```

```
{ int i, index, x;
```

```
int a[5];
```

```
printf("Enter 5 integers: ");
```

```
for(i = 0; i < 5; i++)
```

```
    scanf("%d", &a[i]);
```

```
printf("Enter x: ");
```

```
scanf("%d", &x);
```

```
index = -1;
```

```
for(i = 0; i < 5; i++)
```

```
    if(a[i] == x)
```

```
        index = i;
```

```
if(index != -1) printf("Index is %d\n", index);
```

```
else printf("Not Found\n");
```

```
return 0;
```

```
}
```

例 7-3 思考(2)

Enter 5 integers: **2 9 8 1 9**

Enter x: **9**

Index is 4

index的作用?

❖ 实例：查找（记住下标）

例 7-4 求最小值

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

```
int main(void)
```

```
{ int i, min, n;
```

```
int a[10];
```

```
printf("Enter n: ");
```

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

```
printf("Enter %d integers: ", n);
```

```
for(i = 0; i < n; i++)
```

```
    scanf("%d", &a[i]);
```

```
min = a[0];
```

```
for(i = 1; i < n; i++)
```

```
    if(a[i] < min) min = a[i];
```

```
printf("min is %d \n", min);
```

```
return 0;
```

```
}
```

Enter n: 6

Enter 6 integers: 2 9 -1 8 1 6

min is -1



方法

虽得到了最小值，但不能确定最小值所在下标。

❖ 实例：求最小值

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

```
int main(void)
```

```
{   int i, index, min, n;
```

```
    int a[10];
```

```
    printf("Enter n: ");
```

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

```
    printf("Enter %d integers: ", n);
```

```
    for(i = 0; i < n; i++)
```

```
        scanf("%d", &a[i]);
```

```
    min = a[0]; index = 0;
```

```
    for(i = 1; i < n; i++)
```

```
        if(a[i] < min){
```

```
            min = a[i]; index = i;
```

```
        }
```

```
    printf("min is %d index is %d\n", min, index);
```

```
    return 0;
```

```
}
```

例7-4 求最小值及下标

Enter n: 6

Enter 6 integers: 2 9 -1 8 1 6

min is -1 index is 2

min: 最小值

index: 最小值所在下标

❖ 实例：求最小值（记下标）

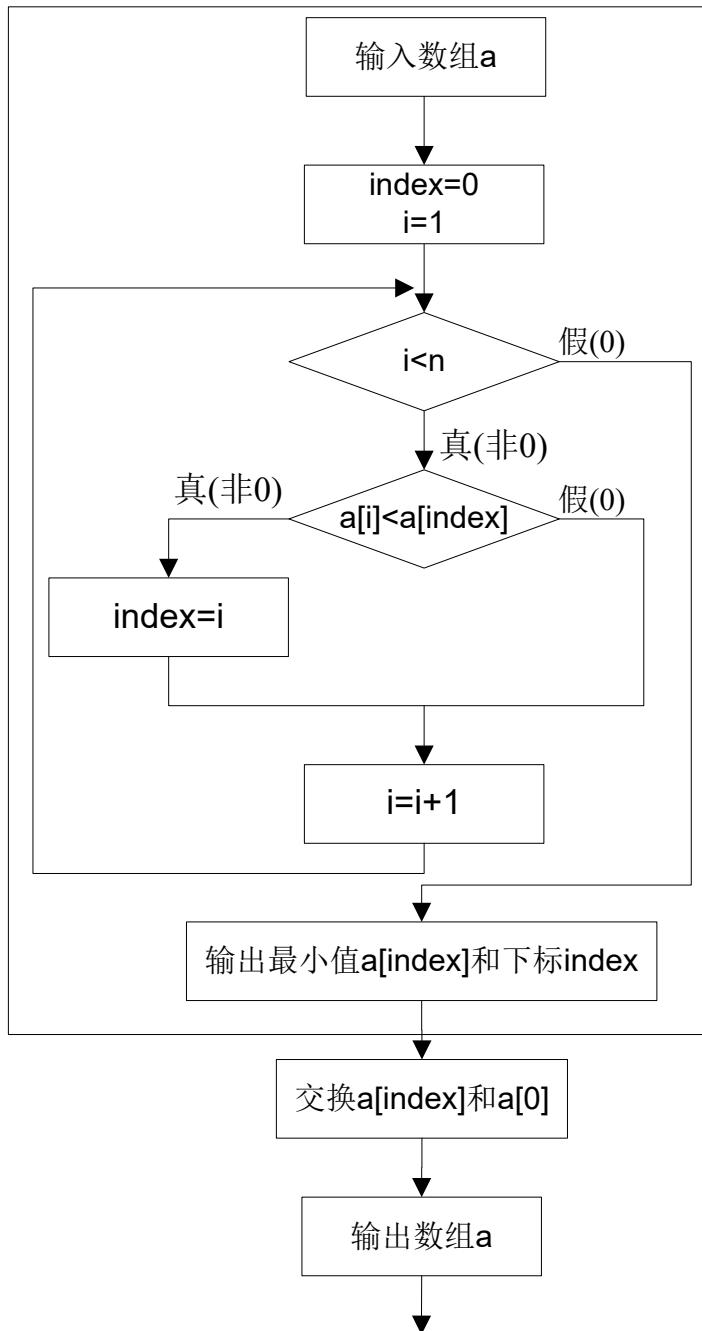
例 7-4(1) 求最小值及其下标

输入 $n(n \leq 10)$, 再输入 n 个数, 输出最小值和它所对应的下标。

用**index**记录最小值对应的下标

a[index]就是最小值

流程图



❖ 实例：求最小值（记下标）

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

```
int main(void)
```

```
{ int i, index, n;
```

```
int a[10];
```

```
printf("Enter n: ");
```

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

```
printf("Enter %d integers: ", n);
```

```
for(i = 0; i < n; i++)
```

```
    scanf("%d", &a[i]);
```

```
index = 0;
```

```
for(i = 1; i < n; i++)
```

```
    if(a[i] < a[index]) index = i;
```

```
printf("min is %d\tindex is %d\n", a[index], index);
```

```
return 0;
```

```
}
```

求最小值及下标

Enter n: 6

Enter 6 integers: 2 9 -1 8 1 6

min is -1 index is 2

❖ 实例：求最小值（仅记下标）

例 7-4(2) 交换最小值

输入 $n(n < 10)$, 再输入 n 个数, 将最小值与第一个数交换, 输出交换后的 n 个数。

用 index 记录最小值对应的下标

$a[\text{index}]$ 就是最小值

最小值与第一个数交换

$a[\text{index}] \rightleftharpoons a[0]$

例 7-5 选择法排序

输入 $n(n < 10)$ ，再输入 n 个数，用选择法将它们从小到大排序后输出。

设 $n = 5$ ，输入为：3 5 2 8 1

下标	0	1	2	3	4
值	3	5	2	8	1

第0趟： 1 5 2 8 3

第1趟： 1 2 5 8 3

第2趟： 1 2 3 8 5

第3趟： 1 2 3 5 8

❖ 实例：选择法排序

选择法分析(1)

3 5 2 8 1 ($n = 5$)

5个数($a[0] \sim a[4]$)中找最小数, 与 $a[0]$ 交换

(0) 1 5 2 8 3 $a[4] \leq \Rightarrow a[0]$

4个数($a[1] \sim a[4]$)中找最小数, 与 $a[1]$ 交换

(1) 1 2 5 8 3 $a[2] \leq \Rightarrow a[1]$

3个数($a[2] \sim a[4]$)中找最小数, 与 $a[2]$ 交换

(2) 1 2 3 8 5 $a[4] \leq \Rightarrow a[2]$

2个数($a[3] \sim a[4]$)中找最小数, 与 $a[3]$ 交换

(3) 1 2 3 5 8 $a[4] \leq \Rightarrow a[3]$

n个数重复n-1次

选择法分析(2)

(0) n 个数 ($a[0] \sim a[n-1]$) 中找最小数, 与 $a[0]$ 交换

(1) n-1个数 ($a[1] \sim a[n-1]$) 中找最小数, 与 $a[1]$ 交换

.....

(k) n-k个数 ($a[k] \sim a[n-1]$) 中找最小数, 与 $a[k]$ 交换

.....

(n-2) 2个数 ($a[n-2] \sim a[n-1]$) 中找最小数, 与 $a[n-2]$ 交换

(0) 5个数 ($a[0] \sim a[4]$) 中找最小数, 与 $a[0]$ 交换

(1) 4个数 ($a[1] \sim a[4]$) 中找最小数, 与 $a[1]$ 交换

(2) 3个数 ($a[2] \sim a[4]$) 中找最小数, 与 $a[2]$ 交换

(3) 4个数 ($a[3] \sim a[4]$) 中找最小数, 与 $a[3]$ 交换

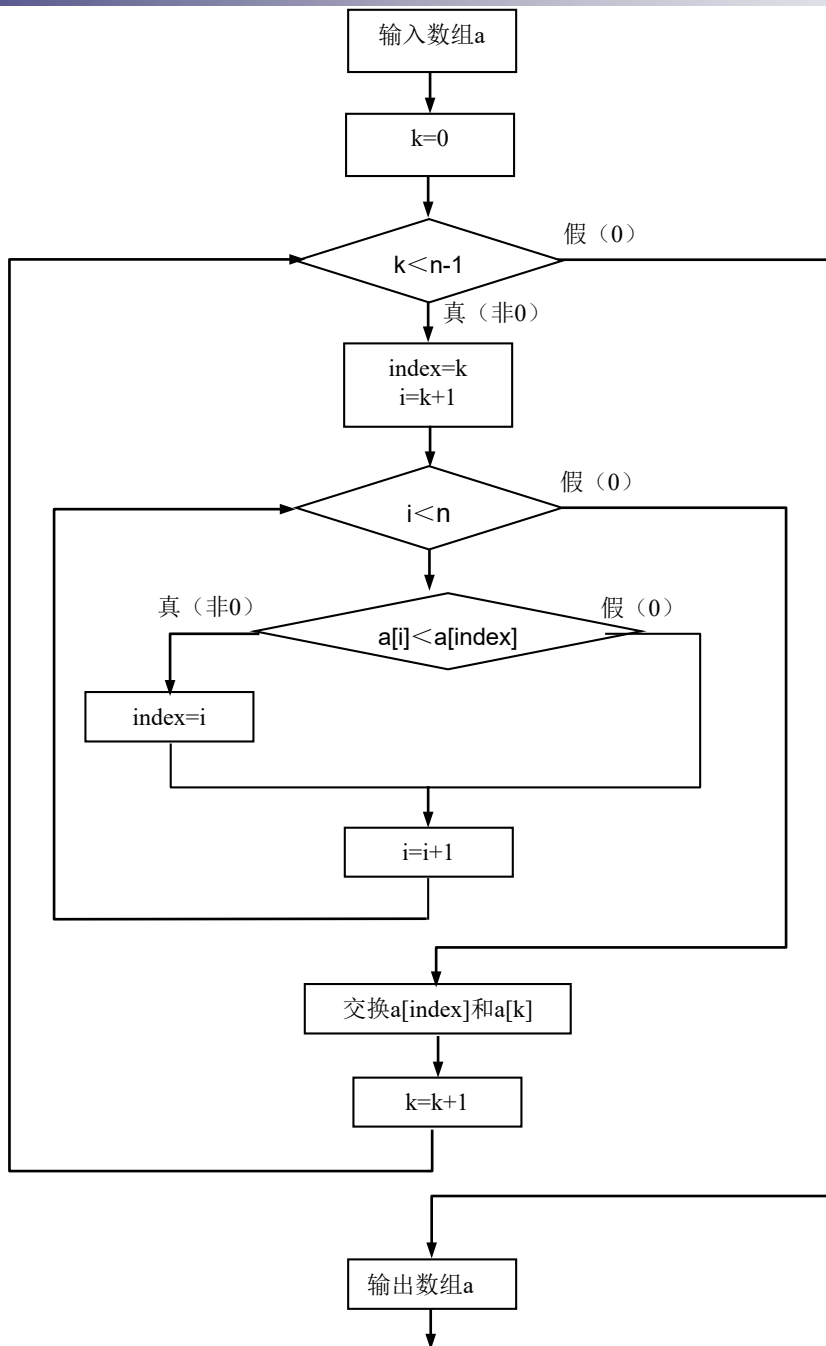
流程图

外循环控制:

n 个数选择
排序共需要
 $n-1$ 次

内循环控制:

在下标范围
 $[k, n-1]$ 内找
最小值所在
位置 $index$



选择法排序 (程序段)

```
for(k = 0; k < n-1; k++){  
    index = k;  
    for(i = k + 1; i < n; i++)  
        if(a[i] < a[index]) index = i;  
    temp = a[index];  
    a[index] = a[k];  
    a[k] = temp;  
}
```

Enter n: 5

Enter 10 integers: 3 5 2 8 1

After sorted: 1 2 3 5 8

例 7-6 投票情况统计

某电视台要调查观众对该台8个栏目（设相应栏目编号为1~8）的受欢迎情况，共调查了1000位观众。现要求编写程序，输入每一位观众的投票情况（每位观众只能选择一个最喜欢的栏目投票），统计并输出各栏目的得票情况。

数组 **count** 保存各栏目的得票数

count[i]: 记录编号为 i (1~8) 的栏目的得票数

count[i]++: 统计编号为 i (1~8) 的栏目的得票数

例7-6 源程序

```
#include <stdio.h>
int main(void)
{   int i, response;
    static int count[9];
    for ( i = 1; i <= 1000; i++) {
        printf( "Enter your response: " );
        scanf ( "%d", &response );
        if (response >= 1 && response <= 8 )
            count[response]++;
        else
            printf ( "invalid: %d\n", response );
    }
    printf ( "result:\n" );
    for ( i = 1; i <= 8; i++ )
        printf ( "%4d%4d\n", i, count[i] );
    return 0;
}
```

```
input your response: 3
input your response: 6
input your response: 9
this is a bad response: 9
input your response: 8
...
result:
1  2
2  0
3  4
...
```

补充 二分法查找

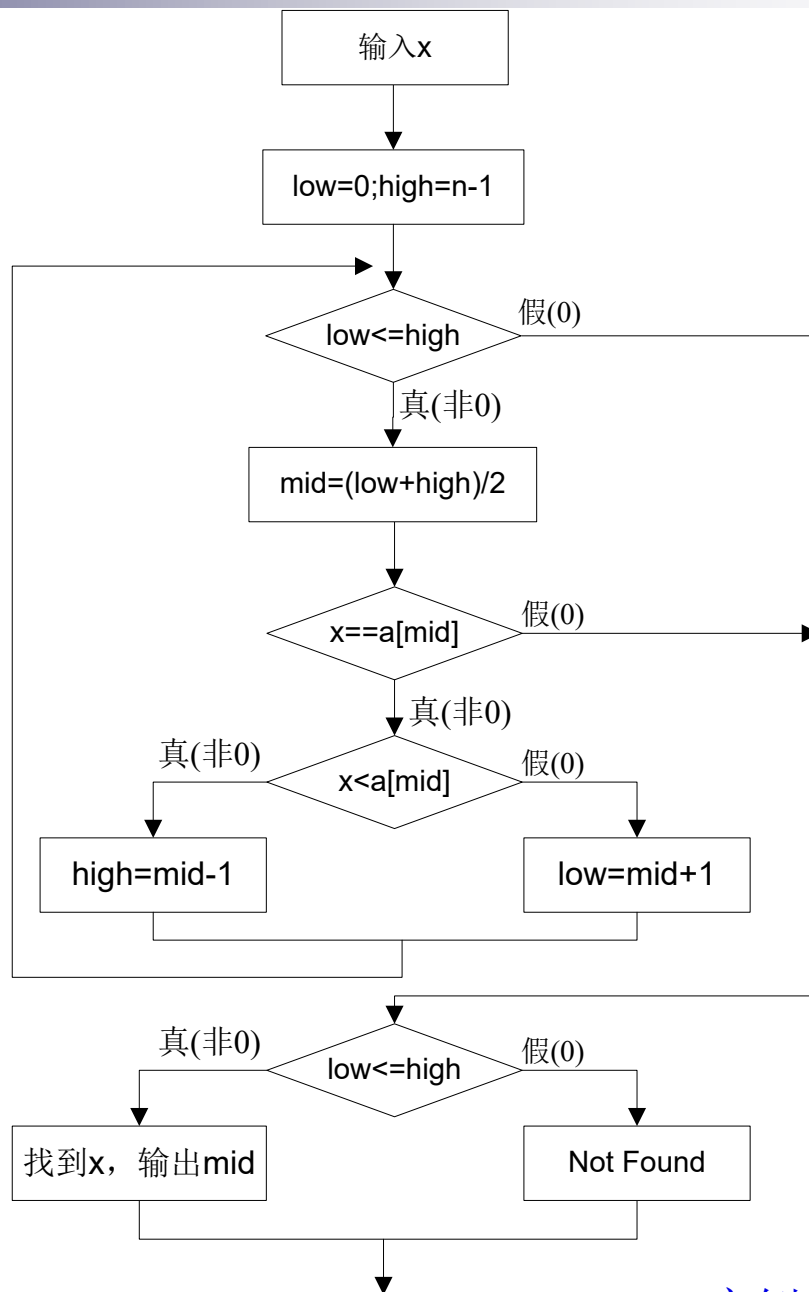
设已有一个**10**个元素的整型数组**a**，且按值从小到大有序排列。

输入一个整数**x**，然后在数组中查找**x**，如果找到，输出相应的下标，否则，输出“**Not Found**”。

例7-3顺序查找算法简单明了，其查找过程就是对数组元素从头到尾的遍历过程。但是，当数组很大时，**查找的效率不高**。

二分查找的效率较高，但前提是**数组元素必须是有顺序的**。

二分法查找流程图



二分法查找 (程序段)

```
low = 0; high = n - 1;          /* 开始时查找区间为整个数组 */
while ( low <= high ) {         /* 循环条件 */
    mid = (low + high) / 2;      /* 中间位置 */
    if ( x == a[mid] )
        break;                  /* 查找成功，中止循环 */
    else if ( x < a[mid] )
        high = mid - 1;          /* 新查找区间为前半段，high前移 */
    else
        low = mid + 1;           /* 新查找区间为后半段，low后移 */
}
if ( low <= high )
    printf("Index is %d \n", mid);
else
    printf( "Not Found\n");
```


7.2 找出矩阵中最大值所在的位置

将1个3*2的矩阵存入1个3*2的二维数组中，找出最大值以及它的行下标和列下标，并输出该矩阵。

7.2.1 程序解析

7.2.2 二维数组的定义和引用

7.2.3 二维数组的初始化

7.2.4 使用二维数组编程

7.2.1 程序解析—求矩阵的最大值

例 7-7 将1个3*2的矩阵存入1个3*2的二维数组中，找出最大值以及它的行下标和列下标，并输出该矩阵。

row 记录最大值的行下标

col 最大值的列下标

a[row][col] 就是最大值

例7-7 源程序

```
int main(void)
{   int col, i, j, row;   int a[3][2];
    printf( "Enter 6 integers:\n" );
    for ( i = 0; i < 3; i++ )
        for ( j = 0; j < 2; j++ )
            scanf ( "%d", &a[i][j] );
    for ( i = 0; i < 3; i++ ){
        for ( j = 0; j < 2; j++ )
            printf("%4d", a[i][j]);
        printf("\n");
    }
    row = col = 0;
    for ( i = 0; i < 3; i++ )
        for ( j = 0; j < 2; j++ )
            if ( a[i][j] > a[row][col] ){
                row = i;   col = j;
            }
    printf ( "max = a[%d][%d] = %d\n", row, col, a[row][col] );
    return 0;
}
```

Enter 6 integers:

3 2

10 -9

6 -1

3 2

10 -9

6 -1

max = a[1][0] = 10

❖ 实例：矩阵的最大值

二维数组

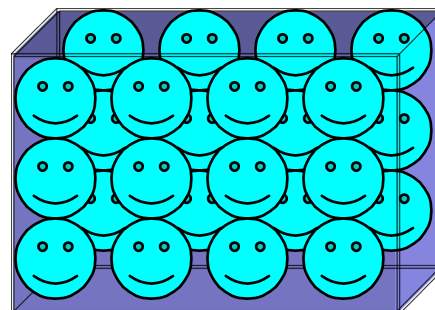
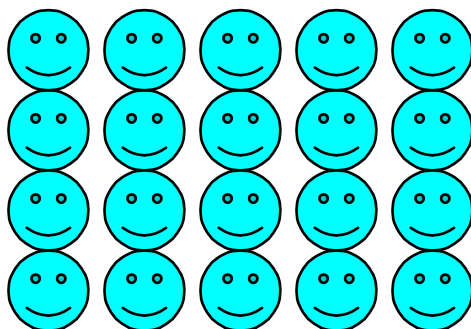
多维数组的空间想象

一维数组： 一列长表或一个向量

二维数组： 一个表格或一个平面矩阵

三维数组： 三维空间的一个方阵

多维数组： 多维空间的一个数据列阵



7.2.2 二维数组的定义和引用

1. 定义

类型名 数组名[行长度][列长度]

int a[3][2];

定义1个二维数组 **a**，3 行 2 列，6 个元素

int b[5][10];

定义1个二维数组 **b**，5 行 10 列，50 个元素

2. 引用

先定义，后使用
数组元素的引用：

数组名[行下标][列下标]

行下标和列下标：整型表达式

行下标的取值范围是[0，行长度-1]

列下标的取值范围是[0，列长度-1]

int a[3][2]; 3 行 2 列， 6个元素

a[0][0] a[0][1]

a[1][0] a[1][1]

a[2][0] a[2][1]

下标不要越界

二维数组在内存中的存放方式

int a[3][2];

3 行 2 列， 6 个元素
表示1个3行2列的矩阵

a[0][0] a[0][1]

a[1][0] a[1][1]

a[2][0] a[2][1]

二维数组的元素在内存中按行/列方式存放

a[0][0]

a[0][1]

a[1][0]

a[1][1]

a[2][0]

a[2][1]

7.2.3 二维数组的初始化

1. 分行赋初值

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

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

数组a

1	2	3
4	5	6
7	8	9

数组b

1	2	3
0	0	0
4	5	0
0	0	0

2. 顺序赋初值

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

```
static int b[4][3] = { 1, 2, 3, 0, 0, 0, 4, 5 };
```


省略行长度

对全部元素都赋了初值

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

或分行赋初值时，在初值表中列出了全部行

```
static int b[ ][3] = { {1, 2, 3}, { }, {4, 5}, { } }
```



建议不要省略

数组a

1	2	3
4	5	6
7	8	9

数组b

1	2	3
0	0	0
4	5	0
0	0	0

7.2.4 使用二维数组编程

行下标和列下标分别做为循环变量，通过二重循环，遍历二维数组

通常将行下标做为外循环的循环变量
列下标 内循环

例7-8 生成一个矩阵并输出

定义1个 **3*2** 的二维数组**a**，数组元素的值由下式给出，按矩阵的形式输出**a**。

$$a[i][j] = i + j \quad (0 \leq i \leq 2, 0 \leq j \leq 1)$$

```
int a[3][2];
```

```
a[0][0]  a[0][1]
```

```
0  1
```

```
a[1][0]  a[1][1]
```

```
1  2
```

```
a[2][0]  a[2][1]
```

```
2  3
```

例7-8 源程序

```
#include <stdio.h>
int main(void)
{   int i, j;
    int a[3][2];

    for ( i = 0; i < 3; i++ )
        for ( j = 0; j < 2; j++ )
            a[i][j] = i + j;

    for ( i = 0; i < 3; i++ ){
        for ( j = 0; j < 2; j++ )
            printf ( "%4d", a[i][j] );
        printf ( "\n" );
    }
    return 0;
}
```

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

i = 0	j = 0
i = 0	j = 1
i = 1	j = 0
i = 1	j = 1
i = 2	j = 0
i = 2	j = 1

二维数组的输入

例7-7中, `int a[3][2];`
`for (i = 0; i < 3; i++)`
 `for (j = 0; j < 2; j++)`
 `scanf ("%d", &a[i][j]);`

Enter 6 integers:

3 2 10 -9 6 -1

3 2

10 -9

6 -1

max = a[1][0] = 10

a[0][0] a[0][1]

a[1][0] a[1][1]

a[2][0] a[2][1]

Enter 6 integers:

3 2 10 -9 6 -1

3 -9

2 6

10 -1

max = a[2][0] = 10

```
for (j = 0; j < 2; j++)  
    for (i = 0; i < 3; i++)  
        scanf("%d", &a[i][j]);
```

矩阵与二维数组

int a[N][N]; N是正整数

a[i][j]: i、j的取值范围 [0, N-1]

用二维数组a表示N*N方阵时，对应关系:

~~a[0][0] a[0][1] a[0][2]~~ 副对角线 **$i+j == N-1$**

~~a[1][0] a[1][1] a[1][2]~~ 上三角 **$i \leq j$**

~~a[2][0] a[2][1] a[2][2]~~ 下三角 **$i \geq j$**

主对角线 **$i == j$**

例7-9 方阵转置

输入一个正整数 n ($1 < n \leq 6$), 根据下式生成1个 $n \times n$ 的方阵, 然后将该方阵转置 (行列互换) 后输出。

$$a[i][j] = i * n + j + 1 \quad (0 \leq i \leq n-1, 0 \leq j \leq n-1)$$

分析: `int a[6][6];` $n=3$ 时

1	2	3		1	4	7
4	5	6	→	2	5	8
7	8	9		3	6	9

`a[i][j]` ↔ `a[j][i]`

`a[0][1]` ↔ `a[1][0]`

`a[0][2]` ↔ `a[2][0]`

`a[1][2]` ↔ `a[2][1]`

❖ 矩阵转置

例7-9 源程序

```
#include <stdio.h>
int main(void)
{   int i, j, n, temp;
    int a[6][6];

    printf ( "Enter n: " );   scanf ( "%d", &n );
    /* 给二维数组赋值 略..... */

    /* 行列互换 */
    for ( i = 0; i < n; i++ )
        for ( j = 0; j < n; j++ )
            if ( i <= j ){          /* 只遍历上三角阵 */
                temp = a[i][j]; a[i][j] = a[j][i]; a[j][i] = temp;
            }

    /* 按矩阵的形式输出a 略..... */
    return 0;
}
```

❖ 矩阵转置

例7-9 说明

/* 行列互换*/

```
for (i = 0; i < n; i++)  
    for (j = 0; j < n; j++)  
        if ( i <= j ) {  
            temp = a[i][j];  
            a[i][j] = a[j][i];  
            a[j][i] = temp;  
        }
```

for (j = i; j < n; j++)

1	2	3
4	5	6
7	8	9

主对角线: $i == j$
上三角: $i \leq j$
下三角: $i \geq j$

	i=0		i=1		
1	4	7	1	4	7
2	5	6	2	5	8
3	8	9	3	6	9

例7-9 思考

```
/* 行列互换 */  
for ( i = 0; i < n; i++ )  
    for ( j = 0; j < n; j++ ){  
        temp = a[i][j];  
        a[i][j] = a[j][i];  
        a[j][i] = temp;  
    }
```

1	2	3
4	5	6
7	8	9

i=0

1	4	7
2	5	6
3	8	9

i=1

1	2	7
4	5	8
3	6	9

i=2

1	2	3
4	5	6
7	8	9

例7-10 日期计算

自定义函数**day_of_year (year, month, day)**，计算并返回年**year**、月**month**和日**day**对应的是该年的第几天。

day_of_year (2000, 3, 1) 返回61

day_of_year (1981, 3, 1) 返回60

分析：

月	0	1	2	3	11	12
非闰年	0	31	28	31		30	31
闰年	0	31	29	31		30	31

```
int tab[2][13] = {  
    { 0, 31, 28, 31, 30,31,30,31,31,30,31, 30,31 }  
    { 0, 31, 29, 31, 30,31,30,31,31,30,31, 30,31 }  
}
```

例7-10 源程序

```
int day_of_year(int year, int month, int day)
{   int k, leap;
    int tab[2][13] = {
        { 0, 31, 28, 31, 30,31,30,31,31,30,31, 30,31 }
        { 0, 31, 29, 31, 30,31,30,31,31,30,31, 30,31 }
    };

    leap = (year%4==0 && year%100!=0) || year %400==0;
    for ( k=1; k<month; k++ )
        day = day + tab[leap][k];

    return day;
}
```

7.3 判断回文

例7-11 输入一个以回车符为结束标志的字符串（少于**80**个字符），判断该字符串是否为回文。

回文就是字符串中心对称，如“**abcba**”
“**abccba**”是回文，“**abcdba**”不是回文。

7.3.1 程序解析

7.3.2 一维字符数组

7.3.3 字符串

7.3.4 使用字符串编程

7.3.1 程序解析- 判断回文

```
int main ( void )
{ int i, k;
  char line[80];
  printf ( "Enter a string: " );
  k = 0;
  while ( (line[k] = getchar() ) != '\n' )
    k++;
  line[k] = '\0';
  i = 0;      /* i是字符串首字符的下标 */
  k = k - 1;  /* k是字符串尾字符的下标 */
  while ( i < k ){
    if ( line[i] != line[k] )
      break;
    i++;
    k--;
  }
  if( i >= k) printf("It is a plalindrome\n");
  else printf("It is not a plalindrome\n");
  return 0;
}
```

Enter a string: **abcba**
It is a plalindrome

Enter a string: **abcdba**
It is not a plalindrome

7.3.2 一维字符数组

- 字符串的存储和运算可以用一维字符数组实现
- 一维字符数组的定义、引用、初始化与其他类型的一维数组一样。

char str[80];

定义一个含有80个字符型元素的数组str

char t[5]={'H', 'a', 'p', 'p', 'y'};

初始化数组 t

t	H	a	p	p	y
---	---	---	---	---	---

t[0] t[1] t[4]

输出数组 t 的所有元素

for (i = 0; i < 5; i++)

putchar (t[i]);

一维字符数组

```
char t[5] = { 'H', 'a', 'p', 'p', 'y' };
```

```
static char s[6] = { 'H', 'a', 'p', 'p', 'y' };
```

```
static char s[6] = { 'H', 'a', 'p', 'p', 'y', 0 };
```

0代表字符 **'\0'**，也就是ASCII码为 0 的字符

```
static char s[6] = { 'H', 'a', 'p', 'p', 'y', '\0' };
```

t	H	a	p	p	y
---	---	---	---	---	---

t[0] t[1]

t[4]

s	H	a	p	p	y	\0
---	---	---	---	---	---	----

s[0] s[1]

s[5]

7.3.3 字符串

字符串常量

用一对双引号括起来的字符序列

一个字符串结束符 **'\0'**

"Happy"

字符串结束符

6个字符 'H' 'a' 'p' 'p' 'y' **'\0'**

有效字符

字符串的**有效长度**：有效字符的个数

字符串与一维字符数组

字符串：一个特殊的一维字符数组

- 把字符串放入一维字符数组（存储）
- 对字符串的操作 **===>** 对字符数组的操作

1. 字符串的存储—数组初始化

字符串可以存放在一维字符数组中

```
static char s[6] = { 'H', 'a', 'p', 'p', 'y', '\0' };
```

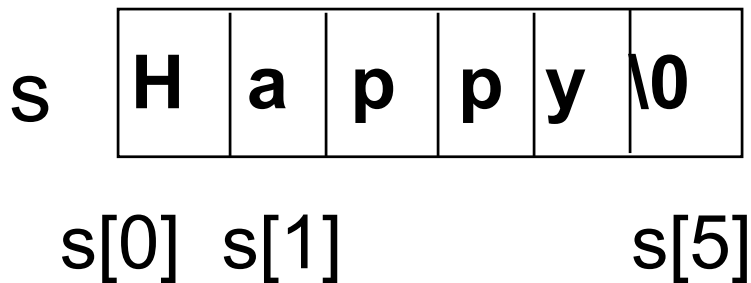
字符数组初始化：用字符串常量

```
static char s[6] = { "Happy" };
```

```
static char s[6] = "Happy";
```

数组长度 \geq 字符串的有效长度 + 1

```
char t[5];      "Happy" 能存入 t 吗?
```



字符串的存储

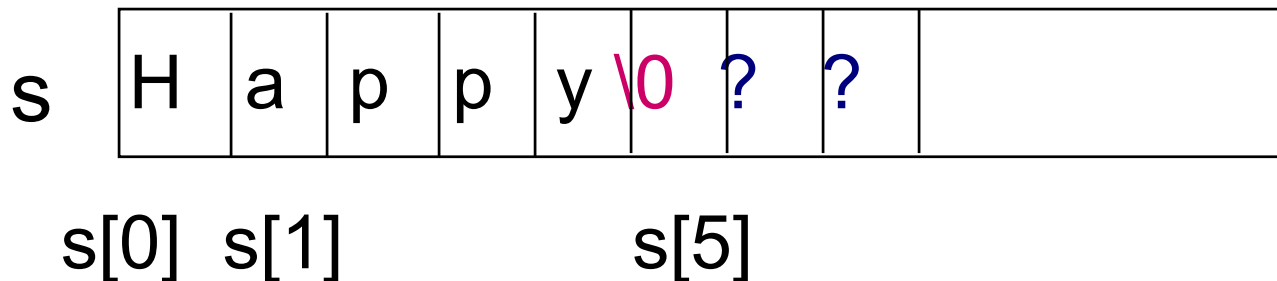
```
auto char s[80] = "Happy";
```

字符串遇 **'\0'** 结束

第一个 **'\0'** 前面的所有字符和 **'\0'** 一起构成了字符串 **"Happy"**

'\0' 之后的其他数组元素与该字符串无关

字符串由有效字符和字符串结束符 **'\0'** 组成



2. 对字符串的操作

- 把字符串放入一维字符数组（存储）
- 对字符串的操作 \Rightarrow 对字符数组的操作
 - 普通字符数组：数组元素的个数是确定的，一般用下标控制循环
 - 字符串：没有显式地给出有效字符的个数，只规定在字符串结束符 `'\0'` 之前的字符都是字符串的有效字符，一般用结束符 `'\0'` 来控制循环
 - 循环条件： `s[i] != '\0'`

输出字符串

```
for ( i = 0; s[i] != '\0'; i++ )
```

输出？

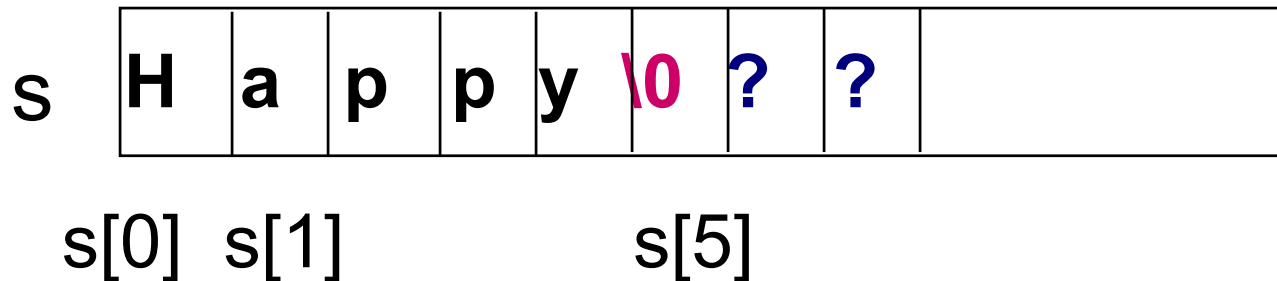
```
    putchar ( s[i] );
```

```
for ( i = 0; i < 80; i++ )
```

```
    putchar ( s[i] );
```

```
for ( i = 0; i < len; i++ )
```

```
    putchar ( s[i] );
```



3. 字符串的存储—赋值和输入

- 把字符串放入一维字符数组（存储）
- 对字符串的操作 ==> 对字符数组的操作
存储

□ 数组初始化

`static char s[6] = "a";`

□ 赋值

`s[0] = 'a'; s[1] = '\0';`

□ 输入

'\0' 代表空操作，无法输入

输入时，设定一个输入结束符

将输入结束符转换为字符串结束符 '\0'

区分"a" 和 'a'

"a" 2 个字符 'a' 和 '\0'

'a' 1 个字符常量

7.3.4 使用字符串编程

C语言将字符串作为一个特殊的一维字符数组来处理。

- 存储：把字符串放入一维字符数组
数组初始化、赋值、输入

对字符串的操作 ==> 对字符数组的操作

- 对一维字符数组的操作：针对字符串的有效字符和字符串结束符
检测字符串结束符 '\0'

例7-12 统计数字字符个数

输入一个以回车符为结束标志的字符串（少于80个字符），统计其中数字字符'0'.....'9'的个数。

分析：

数组长度取上限80

以 '\n' 做为输入结束符

例7-12源程序

```
int main(void)
```

```
{ int count, i;
```

```
char str[80];
```

```
printf ( "Enter a string: " );
```

```
i = 0;
```

```
while ( (str[i] = getchar( )) != '\n' )
```

```
    i++;
```

```
str[i] = '\0'; /* 输入结束符=>字符串结束符 */
```

```
count = 0;
```

```
for ( i = 0; str[i] != '\0'; i++ )
```

```
    if ( str[i] <= '9' && str[i] >= '0' )
```

```
        count++;
```

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

```
return 0;
```

```
}
```

如何改变输入结束符?

字符串的输入

能省略str[i] = '\0'吗?

Enter a string: **It's 512**

count = 3

s

I	t	'	s		5	1	2	\0	?	?
---	---	---	---	--	---	---	---	----	---	---

0 1 2 3 4 5 6 7 8

❖ 字符统计

例7-13 字符串转换

输入一个以回车符为结束标志的字符串（少于**10**个字符），提取其中所有的数字字符（**'0'.....'9'**），将其转换为一个十进制整数输出。

分析：

数组长度取上限**10**

以 **'\n'** 做为输入结束符

"123" ==》 123

s	1	2	3	\0	?	?					
---	---	---	---	----	---	---	--	--	--	--	--

"123" ==》 123

0 1 2 3

n = 0;

for (i = 0; s[i] != '\0'; i++)

if (s[i] <= '9' && s[i] >= '0')

n = n * 10 + (s[i] - '0');

i	s[i]	s[i]-'0'	n = n*10+(s[i]-'0')
0	'1'	1	0*10+1 = 1
1	'2'	2	1*10+2 =12
2	'3'	3	12*10+3 =123
3	'\0'		

例7-13源程序

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

```
int main(void)
```

```
{ int i, number; char s[10];
```

```
printf ( "Enter a string: " ); /* 输入字符串 */
```

```
i = 0;
```

```
while ( (s[i] = getchar( )) != '\n' )
```

```
    i++;
```

```
s[i] = '\0';
```

Enter a string: **a12d3**

digit = 123

a	1	2	d	3	?	?	
0	1	2	3	4			

```
n = 0;
```

```
for ( i = 0; s[i] != '\0'; i++ )
```

```
    if ( s[i] <= '9' && s[i] >= '0' )
```

```
        n = n * 10 + (s[i] - '0');
```

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

```
return 0;
```

```
}
```

a	1	2	d	3	?	?	
0	1	2	3	4			

例7-13 思考

Enter a string: **a12d3**

digit = 123

```

n = 0;
for ( i = 0; s[i] != '\0'; i++ )
    if ( s[i] <= '9' && s[i] >= '0' )
        n = n * 10 + (s[i] - '0');
    else
        break;
digit = ?

```

i	s[i]	s[i]-'0'	n = n*10+(s[i]-'0')
0	'a'		
1	'1'	1	0*10+1 = 1
2	'2'	2	1*10+2 =12
3	'd'		
4	'3'	3	12*10+3 =123
5	'\0'		

例7-14 进制转换

输入一个以'#'为结束标志的字符串（少于**10**个字符），滤去所有的非十六进制字符（不分大小写），组成一个新字符串（十六进制形式），输出该字符串并将其转换为十进制数后输出。

例如：

输入字符串：**zx1?ma0!kbq#**

滤去非十六进制后组成新字符串：**1a0b**

转换为十进制整数：**6667**

例7-14 分析

输入字符串: **zx1?ma0!kbq#**

滤去非十六进制后组成新字符串: **1a0b**

转换为十进制整数: **6667**

分析:

数组长度取上限**10**

以 '#' 做为输入结束符

"zx1?ma0!kbq" ==》 "1a0b"

"1a0b" ==》 6667

生成十六进制字符串

"zx1?ma0!kbq" ==》 "1a0b"

str

hexad

k = 0; /* k: 新字符串hexad的下标 */

for (i = 0; str[i] != '\0'; i++)

if (str[i] >= '0' && str[i] <= '9' || str[i] >= 'a' &&
str[i] <= 'f' || str[i] >= 'A' && str[i] <= 'F'){

hexad[k] = str[i]; /* 放入新字符串 */

k++;

}

hexad[k] = '\0'; /* 新字符串结束标记 */

转换为十进制整数

"1a0b" ==》 6667

hexad number

number = 0; /* 存放十进制数，先清0 */

for (i = 0; hexad[i] != '\0'; i++){ /* 逐个转换 */

if (hexad[i] >= '0' && hexad[i] <= '9')

 number = number * 16 + hexad[i] - '0';

else if (hexad[i] >= 'A' && hexad[i] <= 'F')

 number = number * 16 + hexad[i] - 'A' + 10;

else if (hexad[i] >= 'a' && hexad[i] <= 'f')

 number = number * 16 + hexad[i] - 'a' + 10;

}

输入原字符串
str



滤去非**16**进制
字符后生成新
字符串hexad



把字符串hexad
转换成十进制
整数**number**

```
printf("Enter a string: ");
i = 0;
while ( (str[i] = getchar( )) != '#' )
    i++;
str[i] = '\0';
```

```
k = 0;
for(i = 0; str[i] != '\0'; i++)
    if(str[i]>='0'&&str[i]<='9'
    ||str[i]>='a'&&str[i]<='f' || str[i]>='A'&&str[i]<='F'){
        hexad[k] = str[i];
        k++;
    }
hexad[k] = '\0';
```

```
number = 0;
for(i = 0; hexad[i] != '\0'; i++){
    if(hexad[i] >= '0' && hexad[i] <= '9')
        number = number * 16 + hexad[i] - '0';
    else if(hexad[i] >= 'A' && hexad[i] <= 'F')
        number = number * 16 + hexad[i] - 'A' + 10;
    else if(hexad[i] >= 'a' && hexad[i] <= 'f')
        number = number * 16 + hexad[i] - 'a' + 10;
}
```

字符串小结

字符串：一个特殊的一维字符数组 **'\0'**

■ 把字符串放入一维字符数组（存储）

数组长度足够

- 字符数组初始化： **static char s[80] = "Happy";**
- 赋值： **s[0] = 'a'; s[1] = '\0';**
- 输入： **输入结束符 ==> 字符串结束符 '\0'**

i = 0;

while ((s[i]=getchar()) != '\n')

i++;

s[i] = '\0';

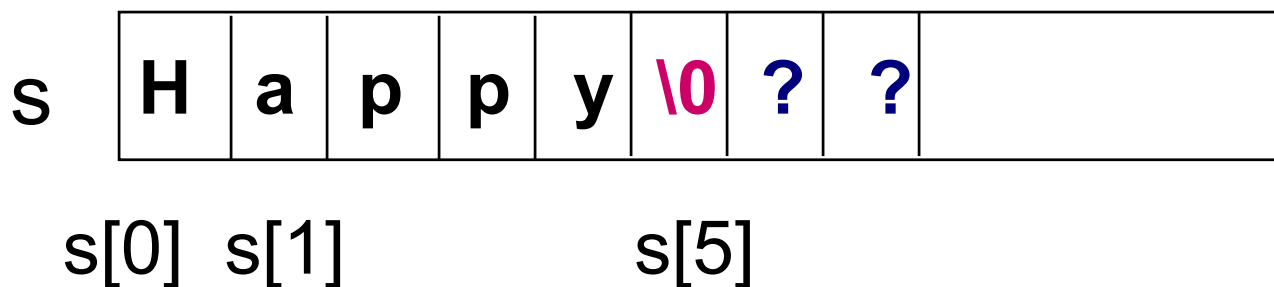
s	H	a	p	p	y	\0	?	?	
---	---	---	---	---	---	----	---	---	--

s[0] s[1]

s[5]

- 把字符串放入一维字符数组（存储）
- 对字符串的操作 **==>** 对字符数组的操作
只针对字符串的有效字符和字符串结束符 **'\0'**

```
for ( i = 0; s[i] != '\0'; i++ )  
    putchar(s[i]);
```



本章总结

■ 一维数组：

- 定义、初始化、引用
- 使用一维数组：选择排

■ 二维数组

- 定义、初始化、引用
- 使用二维数组：矩阵

■ 字符串

- 字符数组与字符串
- 字符串的存储
- 字符串的操作

■ 使用数组进行程序设计

- 正确理解数组的基本概念及在内存中的存放方式；
- 掌握使用一维数组编写程序；
- 掌握使用二维数组编写程序；
- 正确理解字符串的概念，掌握使用字符串编写程序；
- 能合理运用数组进行程序设计，熟练掌握几个常用的算法；

Chap 8 指针

8.1 密码开锁

8.2 角色互换

8.3 冒泡排序

8.4 电码加密

8.5 任意个整数求和*

本章要点

- 变量、内存单元和地址之间是什么关系？
- 如何定义指针变量，怎样才能使用指针变量？
- 什么是指针变量的初始化？
- 指针变量的基本运算有哪些？如何使用指针操作所指向的变量？
- 指针作为函数参数的作用是什么？
- 如何使用指针实现函数调用返回多个值？
- 如何利用指针实现内存的动态分配？

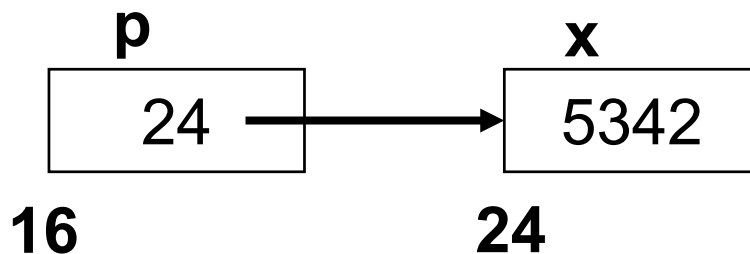
8.1 密码开锁

一个密室逃脱游戏中的密码开锁：

26个寄存箱，每个寄存箱上按顺序都有一个英文字母和一个编号，字母从**A到Z**，**编号从01到26**

关键点分析

- 得到线索：找到一把钥匙，打开**p**寄存箱（假设编号为**16**）
- 提示地址：里面是一把刻着**数字24**的钥匙
- 找到目标：打开编号为**24**的**x**寄存箱
- 取出内容：“**5342**”

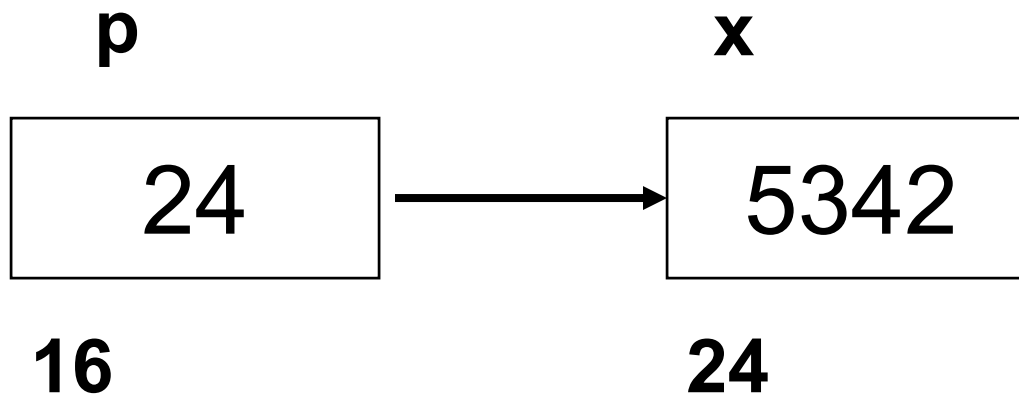


8.1.1 程序解析

寻找密码的途径分析

- 密码存放需要一定的存储空间作为存放地，每个存放地都会有地址
- 如果知道了存放地的名字，当然能够找到密码
- 如果不知道存放地的名字，知道该存放地的地址也能够取出密码
- 如果有另外一个地方存放了该密码存放地的地址，那么就能顺藤摸瓜，间接找到密码

密码存放示意图



名字	p	x
地址	16	24
内容	24	5342

例8-1 利用指针模拟密码开锁游戏

获取密码的两种方法：

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

```
int main(void)
```

```
{
```

```
    int x = 5342;    /* 变量x用于存放密码值5342 */
```

```
    int *p = NULL; /* 定义整型指针变量p，NULL值为0，代表空指针 */
```

```
    p = &x;    /*将变量x的地址存储在p中 */
```

```
    /* 通过变量名x输出密码值*/
```

```
    printf("If I know the name of the variable, I can get it's value by name:  
    %d\n ", x);
```

```
    /* 通过变量x的地址输出密码值 */
```

```
    printf("If I know the address of the variable is: %x, then I also can get it's  
    value by address: %d\n",p, *p);
```

```
    return 0;
```

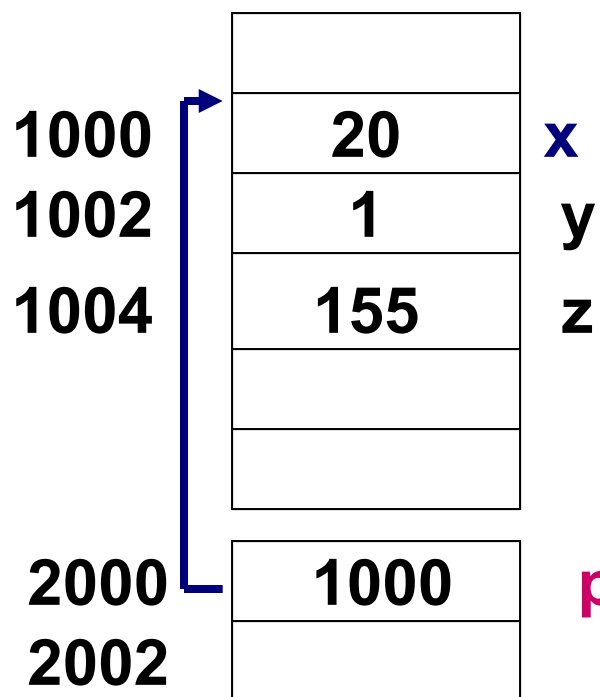
```
}
```

If I know the name of the variable, I can get
it's value by name: **5342**

If I know the address of the variable
is: **12ff7c**, then I also can get it's value by
address: **5342**

8.1.2 地址和指针—指针的概念

内存单元
地址 内容 变量



直接访问：通过变量名访问

```
int x = 20, y = 1, z = 155;  
printf("%d", x);
```

间接访问：通过另一个变量访问
把变量的地址放到另一变量中
使用时先找到后者
再从中取出前者的地址

地址 指针变量

指针

内存单元
地址 内容 变量

1000	20	
1002	1	
1004	155	
2000	1000	
2002		

地址 指针变量

```
int x = 20, y = 1, z = 155;  
printf("%d", x);
```

x
y
z
p

某个变量的地址

指向

指针变量：存放地址的变量

8.1.3 指针变量的定义

类型名 * 指针变量名



指针声明符

指针变量所指向的变量的类型

int *p;

p 是整型指针，指向整型变量

float *fp;

fp 是浮点型指针，指向浮点型变量

char *cp;

cp 是字符型指针，指向字符型变量

指针变量的定义

类型名 * 指针变量名

int * p;

- 指针变量名是 **p**，不是 ***p**
- ***** 是指针声明符

int k, *p1, *p2;

等价于：

int k;

int *p1;

int *p2;

- 定义多个指针变量时，每一个指针变量前面都必须加上*

8.1.4 指针的基本运算

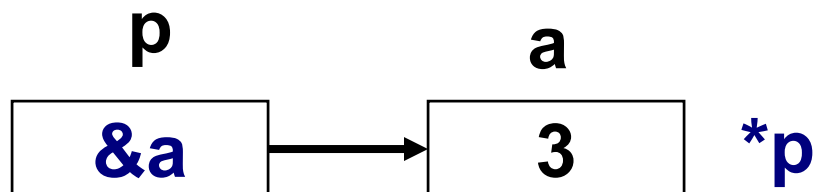
如果指针的值是某个变量的地址，通过指针就能**间接访问**那个变量。

1. 取地址运算和间接访问运算

& 取地址运算符

int *p, a = 3; 指针变量的类型和它所指向变量的类型相同

p = &a; 把 a 的地址赋给 p，即 p 指向 a



***** 指针取值运算符：间接访问运算符，

printf("%d", *p); 输出 p 指向的变量 a 的值

例8-2 指针取地址运算和间接访问运算

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

```
int main (void)
```

```
{ int a = 3, *p;
```

```
  p = &a;
```

```
  printf ("a=%d, *p=%d\n", a, *p);
```

```
  *p = 10;
```

```
  printf("a=%d, *p=%d\n", a, *p);
```

```
  printf("Enter a: ");
```

```
  scanf("%d", &a);
```

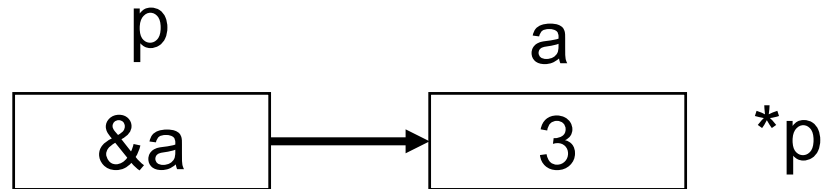
```
  printf("a=%d, *p=%d\n", a, *p);
```

```
  (*p)++;
```

```
  printf("a=%d, *p=%d\n", a, *p);
```

```
  return 0;
```

```
}
```



a = 3, *p = 3

a = 10, *p = 10

Enter a: 5

a = 5, *p = 5

a = 6, *p = 6

说明

(1) 当 $p = \&a$ 后, $*p$ 与 a 相同

(2) `int *p;` 定义指针变量 p

`*p = 10;` 指针 p 所指向的变量, 即 a

(3) `&*p` 与 `&a` 相同, 是地址

`*&a` 与 a 相同, 是变量

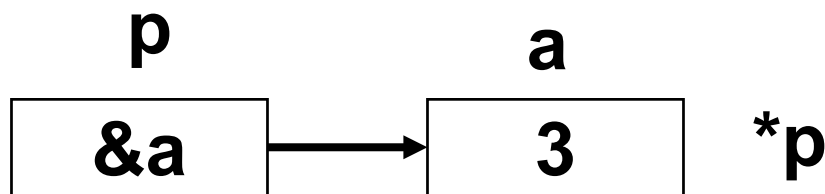
(4) `(*p)++` 等价于 `a++`

将 p 所指向的变量值加1

`*p++` 等价于 `*(p++)`

先取 $*p$, 然后 p 自加, 此时 p 不再指向 a

```
int a = 1, x, *p;  
p = &a;  
printf("%x\n", p);  
x = *p++;  
printf("%d,%x", x, p);
```



2. 赋值运算

```
int a = 3, *p1, *p2;
```

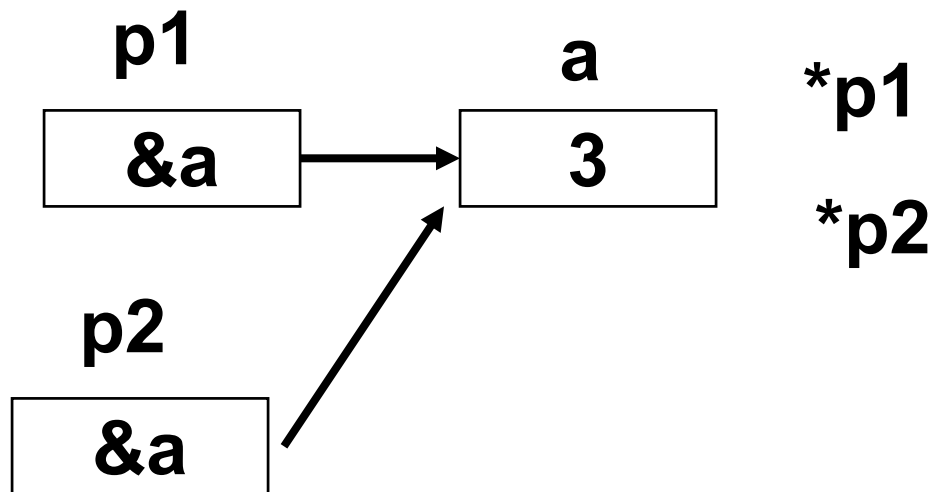
```
p1 = &a;
```

把 a 的地址赋给 p1, 即 p1 指向 a

```
p2 = p1;
```

p2 也指向 a

相同类型的指针才能相互赋值



8.1.5 指针变量的初始化

- 1) 指针变量在定义后也要先赋值再引用
- 2) 在定义指针变量时，可以同时对它赋初值

```
int a;
```

```
int *p1 = &a;
```

```
int *p2 = p1;
```

- 3) 不能用数值作为指针变量的初值，但可以将一个指针变量初始化为一个空指针

```
int *p=1000;
```

```
p = 0;
```

```
p = NULL;
```

```
p = (int*)1732;
```

使用强制类型转换 (int*)
来避免编译错误，不提倡

8.2 角色互换

如何通过函数调用实现代表2个角色的变量互相

...

三套方案

- **swap1()**
- **swap2()**
- **swap3()**

哪个方案能成功？

例8-3 指针作为函数参数模拟角色互换

```
int main (void)
{   int a = 1, b = 2;
    int *pa = &a, *pb = &b;
    void swap1(int x, int y), swap2( int *px, int *py ), swap3 (int *px, int *py);
    swap1 (a, b);
    printf ("After calling swap1: a=%d b=%d\n", a, b);
    a = 1;  b = 2;
    swap2 (pa, pb);
    printf ("After calling swap2: a=%d b=%d\n", a, b);
    a = 1;  b = 2;
    swap3 (pa, pb);
    printf ("After calling swap3: a=%d b=%d\n", a, b);
    return 0;
}
```

调用哪个函数，可以交换main ()
中变量a和b的值？

指针作为函数参数，返回数据

例8-3 swap1()

swap1 (a, b);

```
void swap1 (int x, int y)  
{ int t;  
  
    t = x;  
    x = y;  
    y = t;  
}
```


例8-3 swap2()

```
swap2 (&a, &b);
```

```
void swap2 (int *px, int *py)
{   int t;

    t = *px;
    *px = *py;
    *py = t;
}
```

例8-3 swap3()

swap3 (&a, &b);

void swap3 (int *px, int *py)

{ int *pt;

pt = px;

px = py;

py = pt;

}

After calling swap1: a=1, b=2

After calling swap2: a=2, b=1

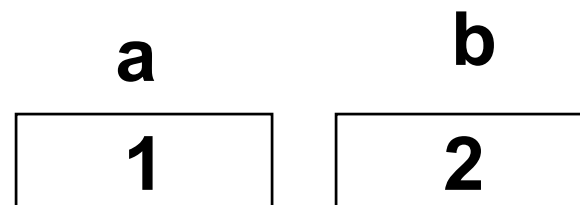
After calling swap3: a=1, b=2

8.2.2 指针作为函数参数

- 函数参数包括**实参**和**形参**，两者的类型要一致，可以是指针类型
- 如果实参是某个变量的地址，相应的形参就是指针
- 在C语言中实参和形参之间的数据传递是**单向**的“**值传递**”方式

例8-3 swap1()

swap1 (a, b);



void swap1 (int x, int y)

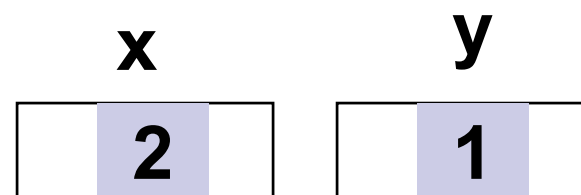
{ int t;

t = x;

x = y;

y = t;

}



在**swap1()**函数中改变了形参**x,y**的值
但不会反过来影响到实参的值

swap1()不能改变**main()**中实参**a**和**b**的值

例8-3 swap2()

```
swap2 (&a, &b);
```

```
void swap2 (int *px, int *py)
```

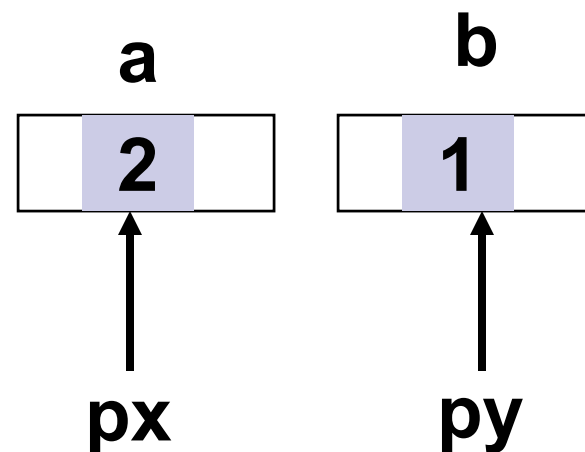
```
{  int t;
```

```
    t = *px;
```

```
    *px = *py;
```

```
    *py = t;
```

```
}
```



在swap2()函数中交换*px和*py的值，主调函数中a和b的值也相应交换了

**值传递，地址未变，
但存放的变量值改变了**

指针作为函数参数，返回数据

例8-3 swap3()

```
swap3 (&a, &b);
```

```
void swap3 (int *px, int *py)
```

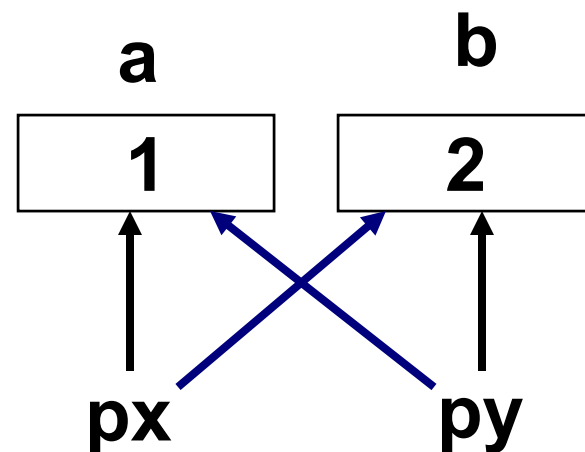
```
{  int *pt;
```

```
    pt = px;
```

```
    px = py;
```

```
    py = pt;
```

```
}
```



swap3()中直接交换了形参指针px和py的值

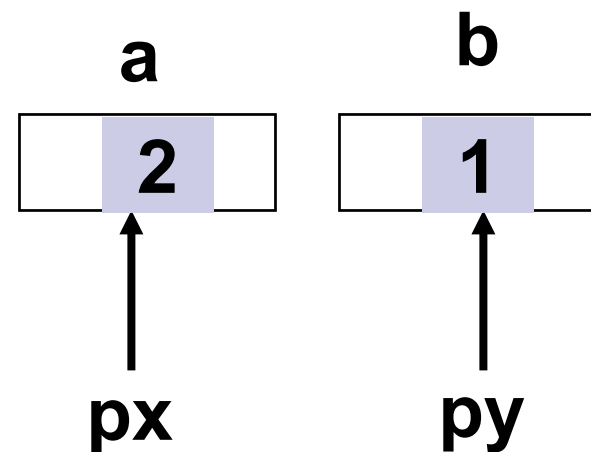
**值传递，形参指针的改变
不会影响实参**

指针作为函数参数的应用

```
swap2 (&a, &b);
```

```
void swap2 (int *px, int *py)
```

```
{  int t;  
    t = *px;  
    *px = *py;  
    *py = t;  
}
```



要通过函数调用来改变主调函数中某个变量的值：

- (1) 在主调函数中，将该变量的地址或者指向该变量的指针作为实参
- (2) 在被调函数中，用指针类型形参接受该变量的地址
- (3) 在被调函数中，改变形参所指向变量的值

通过指针实现函数调用返回多个值

例8-4 输入年和天数，输出对应的年、月、日。

例如：输入2000和61，输出2000-3-1。

定义函数 `month_day(year, yearday, *pmonth, *pday)`

用2个指针作为函数的参数，带回2个结果

```
int main (void)
```

```
{
```

```
    int day, month, year, yearday;
```

```
    void month_day(int year, int yearday, int *pmonth, int *pday);
```

```
    printf ("input year and yearday: ");
```

```
    scanf ("%d%d", &year, &yearday );
```

```
    month_day (year, yearday, &month, &day );
```

```
    printf ("%d-%d-%d \n", year, month, day );
```

```
    return 0;
```

```
}
```

指针作为函数参数，返回多个数据

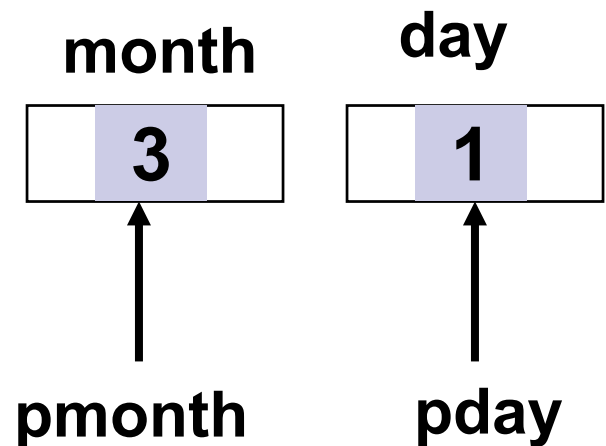
例8-4

```
void month_day ( int year, int yearday, int * pmonth, int * pday)
{ int k, leap;
  int tab [2][13] = {
    {0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 },
    {0, 31, 29, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 },
  };
  /* 建立闰年判别条件leap */
  leap = (year%4 == 0 && year%100 != 0) || year%400 == 0;

  for ( k = 1; yearday > tab[leap][k]; k++)
    yearday = yearday-tab [leap][k];

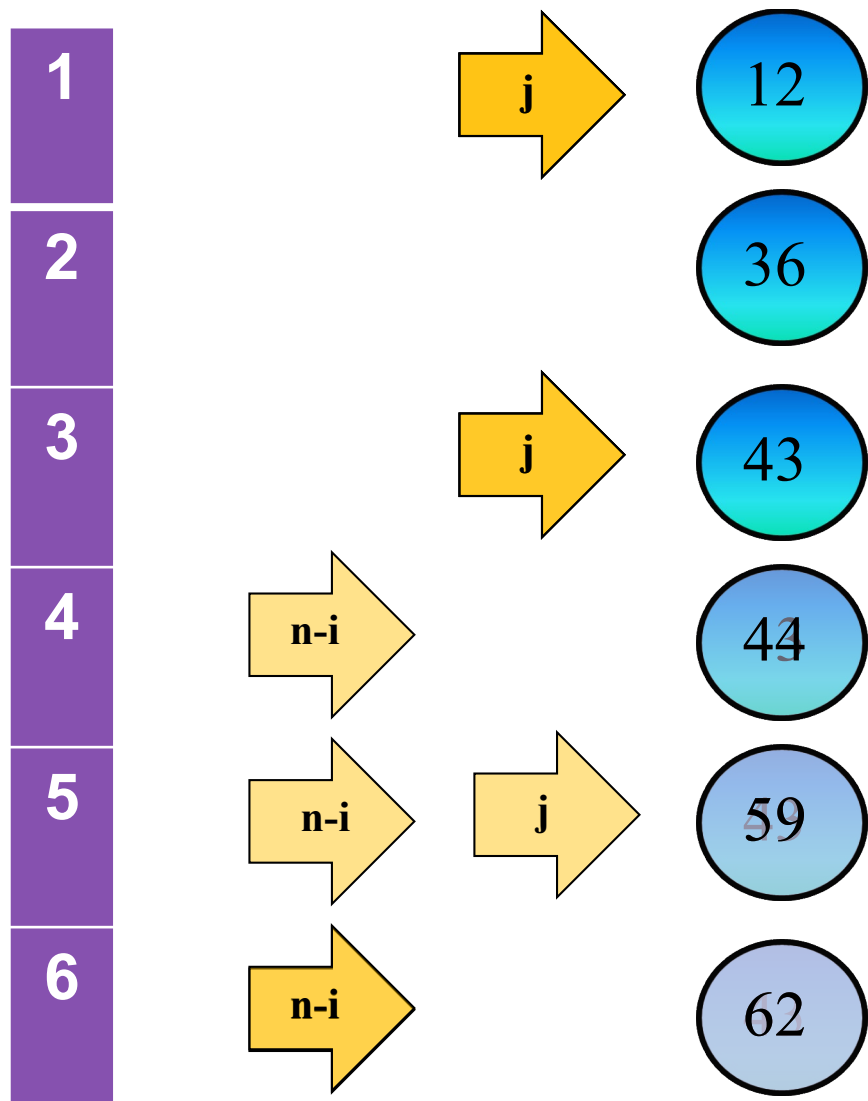
  *pmonth = k;
  *pday = yearday;
}
```

input year and yearday: 2000 61
2000-3-1



指针作为函数参数，返回多个数据

8.3 冒泡排序算法分析



8.3.1 程序解析

(以先定位大的数为例)

```
void bubble (int a[ ], int n)
{   int i, j;
    for( i = 1; i < n; i++ )
        for (j = 0; j < n-i; j++ )
            if (a[j] > a[j+1]){
                t=a[j]; a[j]=a[j+1]; a[j+1]=t;
            }
}
```

```
int main (void)
{   int n, a[8]; int i;
    /* 输入a[ ]; */
    bubble(a, n);
    /* 输出 a[ ]; */
    return 0;
}
```

数组名作为函数参数，修改数组元素内容

8.3.2 数组和地址间的关系

```
int a[100], *p;
```

数组名代表一个地址，它的值是数组首元素的地址（基地址）

a+i 是数组a的基地址的第i个偏移量

	地址	内容	数组元素
a	3000		a[0]
a+1	3002		a[1]
a+i			a[i]
a+99	3198		a[99]

&a[i] ***(a+i)**

```
sum = 0;
```

```
for(i = 0; i < 100; i++)
```

```
    sum = sum + *(a+i) ;
```

下标运算符[]的含义

数组名—地址常量

指针和数组的关系

任何由数组下标来实现的操作

都能用指针来完成

```
int a[100], *p;
```

```
p = a;
```

或

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

p **a**

p+1 **a+1**

p+i **a+i**

p+99 **a+99**

地址 内容 数组元素

3000

 a[0]

3002

 a[1]

 a[i]

 a[99]

&a[i]

a+i

p+i

&p[i]

等价

a[i]
*(a+i)

等价

*(p+i)
p[i]

```
p = a;
```

```
sum = 0;
```

```
for (i = 0; i < 100; i++)
```

```
    sum = sum + p[i];
```

用指针完成对数组的操作

```
int a[100], *p;
```

移动指针

		地址	内容	数组元素
p	a	3000		a[0]
p	a+1	3002		a[1]
p	a+i			a[i]
p	a+99	3198		a[99]

```
sum = 0;
```

```
for (p = a; p <= &a[99]; p++)
```

```
    sum = sum + *p;
```

例8-6 使用指针计算数组元素个数和数组元素的存储单元数

		地址	内容	数组元素
<pre># include <stdio.h> int main (void) { double a[2], *p, *q; p = &a[0]; q = p + 1; printf ("%d\n", q - p); printf ("%d\n", (int) q - (int) p); return 0; }</pre>	p	a	3000	a[0]
	q	a+1	3008	a[1]
		指针p和q之间元素的个数		
		指针p和q之间的字节数		
		地址值		

1

8

指针的算术运算和比较运算

			地址	内容	数组元素
double *p, *q;	p	a	3000		a[0]
	q	a+1	3008		a[1]

■ $q - p$

两个相同类型的指针相减，表示它们之间相隔的存储单元的数目

■ $p + 1 / p - 1$

指向下一个存储单元 / 指向上一个存储单元

■ 其他操作都是非法的

指针相加、相乘和相除，或指针加上和减去一个浮点数

■ $p < q$

两个相同类型指针可以用关系运算符比较大小

例8-7 分别使用数组和指针计算数组元素之和

```
int main(void)
{   int i, a[10], *p;
    long sum = 0;
    printf ("Enter 10 integers: ");
    for (i = 0; i < 10; i++)
        scanf ("%d", &a[i]);
    for ( i = 0; i < 10; i++)
        sum = sum + a[i];
    printf ("calculated by array,
        sum=%ld \n", sum);

    sum=0;
    for (p = a; p <= a+9; p++)
        sum = sum + *p;
    printf ("calculated by pointer,
        sum=%ld \n", sum);
    return 0;
}
```

		地址	内容	数组元素
p	a	3000		a[0]
p	a+1	3002		a[1]
p	a+i			a[i]
p	a+9	3018		a[9]

Enter 10 integers: **10 9 8 7 6 5 4**
3 2 1
calculated by array, sum=**55**
calculated by pointer, sum=**55**

8.3.3 数组名作为函数的参数

数组元素作为函数实参时，函数形参为变量
与变量作为函数实参相同，值传递

```
double fact (int n);
int main(void )
{
    int a[5]={1, 4, 5, 7, 9};
    int i, n = 5;
    double sum;
    sum = 0;
    for(i = 1; i <= n; i++ )
        sum = sum + fact(a[i-1]);
    printf("sum = %e\n", sum);
    return 0;
}
```

```
double fact (int n)
{
    int i;
    double result = 1;
    for (i = 1; i <= n; i++)
        result = result * i ;
    return result ;
}
```

1!+4!+5!+7!+9!

数组元素作为函数参数

数组名作为函数的参数

- 数组名是指针常量，相当于指针作为函数的参数
- 数组名做为实参，形参是指针变量（数组）

(1) 实参是数组名

(2) 形参是指针变量

可以写成数组形式

`int a[]`

```
int sum (int *a, int n)
{   int i, s = 0;
    for(i=0; i<n; i++)
        s += a[i];
        *(a+i)
    return(s);
}
```

例

```
int main(void )
{   int i;
    int b[5] = {1, 4, 5, 7, 9};
    printf("%d\n", sum(b, 5));
    return 0;
}
```

数组名作为函数参数

```

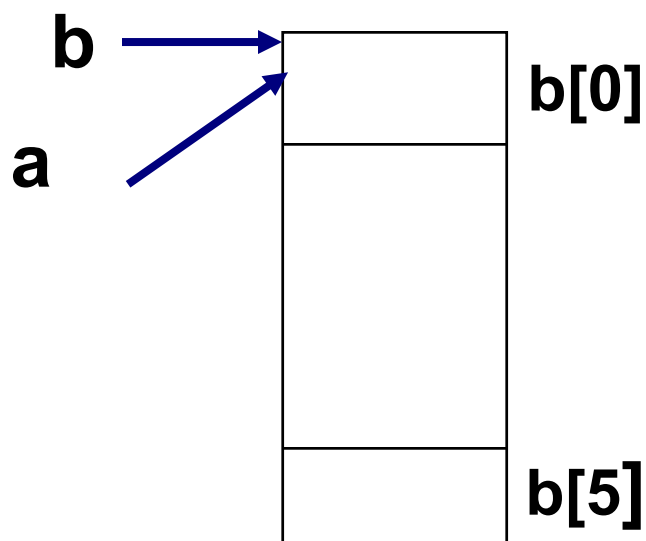
int sum (int *a, int n)
{
    int i, s = 0;
    for (i = 0; i < n; i++)
        s += a[i];
    return (s);
}

```

```

int main (void )
{
    int i;
    int b[5] = {1, 4, 5, 7, 9};
    printf ("%d\n", sum(b, 5));
    return 0;
}

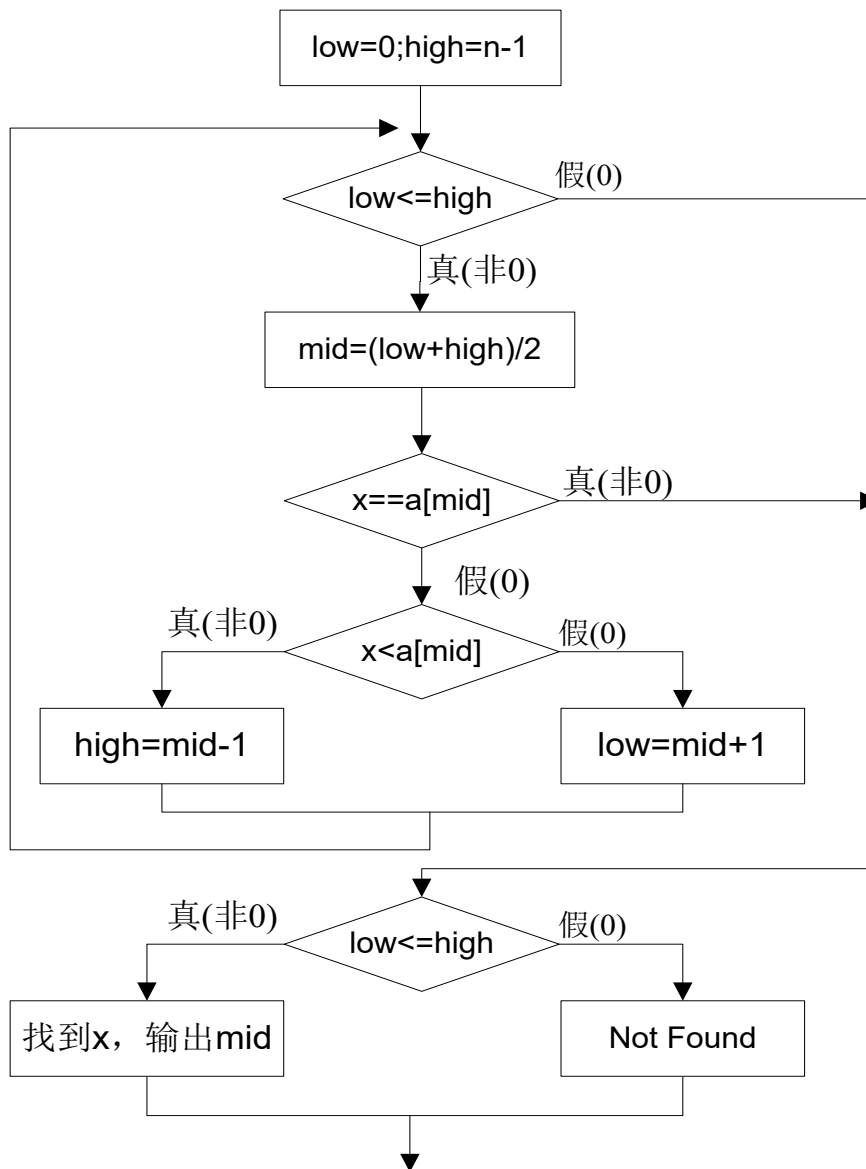
```



<code>sum(b, 5)</code>	<code>b[0]+b[1]+...+b[4]</code>
<code>sum(b, 3)</code>	<code>b[0]+b[1]+b[2]</code>
<code>sum(b+1, 3)</code>	<code>b[1]+b[2]+b[3]</code>
<code>sum(&b[2], 3)</code>	<code>b[2]+b[3]+b[4]</code>

数组名作为函数参数

例8-8二分查找

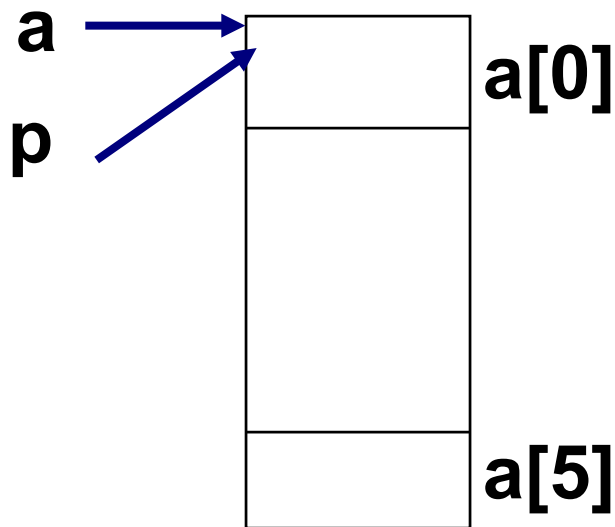


例8-8二分查找

```
int Bsearch(int *p, int n, int x) /* 二分查找函数 */
{
    int low, high, mid;
    low = 0; high = n - 1; /* 开始时查找区间为整个数组 */
    while (low <= high) { /* 循环条件 */
        mid = (low + high) / 2; /* 中间位置 */
        if (x == p[mid])
            break; /* 查找成功，中止循环 */
        else if (x < p[mid]) high = mid - 1; /* 前半段，high前移 */
        else low = mid + 1; /* 后半段，low后移 */
    }
    if (low <= high)
        return mid; /* 找到返回下标 */
    else
        return -1; /* 找不到返回-1 */
}
```

数组名（指针）作为函数参数

- 数组名做为函数的参数，在函数调用时，将**实参数组**首元素的地址传给**形参**（**指针变量**），因此，形参也指向实参数组的首元素。如果改变形参所指向单元的值，就是改变实参数组首元素的值。
- 或：**形参数组和实参数组**共用同一段存储空间，如果形参数组中元素的值发生变化，实参数组中元素的值也同时发生变化。



8.3.4 冒泡排序算法分析

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

```
int main(void )
{
    int i, a[10];
    for(i=0; i<10; i++)
        scanf("%d", &a[i]);

    sort(a, 10);

    for(i=0; i<10; i++)
        printf("%d ", a[i]);
    printf("\n");
    return 0;
}
```

数组名作为函数参数，修改数组元素内容

8.4 电码加密

字符串： 字符数组
 字符指针

8.4.1 程序解析

8.4.2 字符串和字符指针

8.4.3 常用的字符串处理函数

8.4.1 程序解析—加密

```
# define MAXLINE 100
void encrypt(char *);
int main (void)
{ char line [MAXLINE];
```

```
    printf ("Input the string: ");
    gets (line);
    encrypt (line);
```

```
    printf ("%s%s\n", "After being encrypted: ", line);
```

```
    return 0;
```

```
}
```

```
void encrypt ( char *s)
{
    for ( ; *s != '\0'; s++)
        if (*s == 'z')
            *s = 'a';
        else
            *s = *s+1;
}
```

Input the string: **hello hangzhou**

After being encrypted: **ifmmp!ibohaipv**

8.4.2 字符串和字符指针

■ 字符串常量

"array"

"point"

- 用一对双引号括起来的字符序列
- 被看做一个特殊的一维字符数组,在内存中连续存放
- 实质上是一个指向该字符串首字符的指针常量

```
char sa[ ] = "array";
```

```
char *sp = "point";
```

char sa[] = "array";

char *sp = "point";

printf("%s ", sa);

printf("%s ", sp);

printf("%s\n", "string");

printf("%s ", sa+2);

printf("%s ", sp+3);

printf("%s\n", "string"+1);

ray nt tring

array point string

数组名**sa**、指针**sp**和字符串 **"string"** 的值都是
地址

字符数组与字符指针的重要区别

```
char sa[ ] = "This is a string";
```


```
char *sp = "This is a string";
```

sa

T	h	i	s		i	s		a		s	t	r	i	n	g	\0
---	---	---	---	--	---	---	--	---	--	---	---	---	---	---	---	----

sp

--



T	h	i	s		i	s		a		s	t	r	i	n	g	\0
---	---	---	---	--	---	---	--	---	--	---	---	---	---	---	---	----

如果要改变数组**sa**所代表的字符串，只能改变
数组元素的内容

如果要改变指针**sp**所代表的字符串，通常直接
改变指针的值，让它指向新的字符串

示例

```
char sa[ ] = "This is a string";
```

```
char *sp = "This is a string";
```

```
strcpy (sa, "Hello");
```

```
sp = "Hello";
```

```
sa = "Hello"; 非法
```

数组名是常量，不能对它赋值

字符指针—先赋值，后引用

定义字符指针后，如果没有对它赋值，指针的值
不确定。

```
char *s ;
```

```
scanf ("%s", s);
```

不要引用未赋值的指针

```
char *s, str[20];
```

```
s = str;
```

```
scanf ("%s", s);
```

定义指针时，先将它的初值置为空

```
char *s = NULL
```

加密函数的两种实现

```
void encrypt ( char *s)
{
    for ( ; *s != '\0'; s++)
        if (*s == 'z')
            *s = 'a';
        else
            *s = *s+1;
}
```

```
void encrypt (char s[ ])
{
    int i;
    for (i = 0; s[i] != '\0'; i++)
        if (s[i] == 'z')
            s[i] = 'a';
        else
            s[i] = s[i]+1;
}
```


8.4.3 常用的字符串处理函数

- 函数原型在 **stdio.h** 或 **string.h** 中给出

1. 字符串的输入和输出

- 输入字符串: **scanf ()**或**gets ()**
- 输出字符串: **printf ()**或**puts ()**
- **stdio.h**

字符串的输入

```
char str[80];  
i = 0;  
while((str[i] = getchar( )) != '\n')  
    i++;  
str[i] = '\0';
```

(1) scanf("%s", str)

输入参数：字符数组名,不加地址符

遇回车或空格输入结束，并自动将输入的一串字符和 '\0' 送入数组中

(2) gets(str)

遇回车输入结束，自动将输入的一串字符和 '\0' 送入数组中

字符串的输出

```
char str[80];  
for (i = 0; str[i] != '\0'; i++)  
    putchar (str[i]);
```

```
(3) printf ("%s", str)  
    printf ("%s", "hello");
```

```
(4) puts (str)  
    puts ("hello");  
输出字符串后自动换行
```

输出参数可以是字符数组名或字符串常量，输出遇 '\0' 结束

例8-10 字符串输入输出函数示例

```
# include <stdio.h>
int main ( )
{ char str[80];
  scanf ("%s", str);
  printf ("%s", str);
  printf ("%s", "Hello");
  return 0;
}
```

Programming

ProgrammingHello

Programming is fun!

ProgrammingHello

```
# include <stdio.h>
int main ( )
{ char str[80];
  gets (str);
  puts (str);
  puts ("Hello");
  return 0;
}
```

Programming

Programming

Hello

Programming is fun!

Programming is fun!

Hello

2. 字符串的复制、连接、比较、求字符串长度

- 求字符串长度: **strlen (str)**
- 字符串复制: **strcpy (str1, str2)**
- 字符串连接: **strcat (str1, str2)**
- 字符串比较: **strcmp (str1, str2)**
- **string.h**

字符串长度函数strlen

■ strlen (str)

计算字符串的有效长度，不包括 '\0'。

```
static char str[20] = "How are you?"
```

strlen ("hello") 的值是: **5**

strlen (str) 的值是: **12**

字符串复制函数strcpy ()

strcpy (str1, str2);

将字符串 **str2** 复制到 **str1** 中

static char str1[20];

static char str2[20] = "happy";

\0					
----	--	--	--	--	--

h	a	p	p	y	\0
---	---	---	---	---	----

strcpy (str1, str2);

str1 中

h	a	p	p	y	\0
---	---	---	---	---	----

strcpy (str1, "world");

str1 中:

w	o	r	l	d	\0
---	---	---	---	---	----

strcpy () 示例

```
# include "stdio.h"
# include "string.h"
int main (void )
{
    char str1[20], str2[20];
    gets (str2);
    strcpy (str1, str2);
    puts (str1);
    return 0;
}
```

1234

1234

```
char* mystrcpy(char *s1,
char *s2 )
{ char *s=s1;

    return s;
}
```


字符串连接函数strcat

strcat (str1, str2);

连接两个字符串 **str1** 和 **str2**, 并将结果放入 **str1** 中

```
# include "stdio.h"
```

```
# include "string.h"
```

```
int main (void)
```

```
{
```

```
    char str1[80], str2[20];
```

```
    gets (str1);
```

```
    gets (str2);
```

```
    strcpy (str1+strlen(str1), str2);
```

```
    puts (str1);
```

```
    return 0;
```

```
}
```

str1中: Let us \0

str2中: go.\0

str1中: Let us go.\0

str2中: go.\0

Let us
go.
Let us go.

str1=str1+str2 非法!

字符串的拼接

字符串比较函数strcmp

strcmp (str1, str2)

比较两个字符串 **str1** 和 **str2** 的大小。

规则：按字典序(ASCII码序)

- 如果 **str1** 和 **str2** 相等，返回 **0**;
- 如果 **str1** 大于 **str2** ， 返回一个正整数;
- 如果 **str1** 小于 **str2** ， 返回一个负整数;

static char s1[20] = "sea";

strcmp (s1, "Sea");

strcmp ("Sea", "Sea ");

strcmp ("Sea", "Sea");

正整数

负整数

0


strcmp () 示例

```
# include <stdio.h>
# include <string.h>
int main (void)
{
    int res;
    char s1[20], s2[20];
    gets (s1);
    gets (s2);
    res = strcmp (s1, s2);
    printf(“%d\n”, res);
    return 0;
}
```

1234

2

-1

```
int mystrcmp(char *s1,
             char *s2 )
{
    
    return *s1-*s2;
}
```

字符串内容的比较

用strcmp ()比较字符串

利用字符串比较函数比较字符串的大小

strcmp (str1, str2);

为什么定义这样的函数？

str1 > str2

str1 < "hello"

str1 == str2

strcmp (str1, str2) > 0

strcmp (str1, "hello") < 0

strcmp (str1, str2) == 0

比较字符串首元素的地址

比较字符串的内容

字符串处理函数小结

函数	功能	头文件
puts (str)	输出字符串	stdio.h
gets (str)	输入字符串（回车间隔）	
strcpy (s1,s2)	s2 ==> s1	string.h
strcat (s1,s2)	s1 “+” s2 ==> s1	
strcmp(s1,s2)	若 s1“==”s2 ，函数值为 0 若 s1 “>” s2 ，函数值 >0 若 s1 “<” s2 ，函数值 <0	
strlen(str)	计算字符串的有效长度 不包括 '\0'	

例8-11 求最小字符串

```
int main ( )
{ int i;
  int x, min;
  scanf ("%d", &x);
  min = x;
  for (i = 1; i < 5; i++){
    scanf ("%d", &x);
    if (x < min)
      min = x;
  }
  printf ("min is %d\n", min);
  return 0;
}
```

2 8 -1 99 0

min is -1

tool key about zoo sea
min is about

```
#include <string.h>
int main ( )
{ int i;
  char sx[80], smin[80];
  scanf ("%s", sx);
  strcpy (smin,sx);
  for (i = 1; i < 5; i++){
    scanf ("%s", sx);
    if (strcmp (sx, smin)<0)
      strcpy (smin,sx);
  }
  printf ("min is %s\n", smin);
  return 0;
}
```

8.5 任意个整数求和 *

例8-12 先输入一个正整数 n ，再输入任意 n 个整数，计算并输出这 n 个整数的和。

要求使用动态内存分配方法为这 n 个整数分配空间。

8.5.1 程序解析

```
int main ( )
{   int n, sum, i, *p;
    printf ("Enter n: ");
    scanf ("%d", &n);
    if ((p = (int *) calloc (n, sizeof(int))) == NULL) {
        printf ("Not able to allocate memory. \n");
        exit(1);
    }
    printf ("Enter %d integers: ", n);
    for (i = 0; i < n; i++)
        scanf ("%d", p+i);
    sum = 0;
    for (i = 0; i < n; i++)
        sum = sum + *(p+i);
    printf ("The sum is %d \n",sum);
    free (p);
    return 0;
}
```

Enter n: 10

Enter 10 integers: 3 7 12 54 2 -19 8 -1 0 15

The sum is 81

8.5.2 用指针实现内存动态分配

- 变量在使用前必须被定义且安排好存储空间
- 全局变量、静态局部变量的存储是在编译时确定，在程序开始执行前完成。
- 自动变量，在执行进入变量定义所在的复合语句时为它们分配存储，变量的大小也是静态确定的。
- 一般情况下，运行中的很多存储要求在写程序时无法确定。

动态存储管理

- 不是由编译系统分配的，而是由用户在程序中通过动态分配获取。
- 使用动态内存分配能有效地使用内存
 - 使用时申请
 - 用完就释放

同一段内存可以有不同的用途

动态内存分配的步骤

- (1) 了解需要多少内存空间
- (2) 利用**C**语言提供的动态分配函数来分配所需要的存储空间。
- (3) 使指针指向获得的内存空间，以便使用指针在该空间内实施运算或操作。
- (4) 当使用完毕内存后，释放这一空间。

动态存储分配函数malloc()

void *malloc(unsigned size)

在内存的动态存储区中分配一连续空间，其长度为
size

- 若申请成功，则返回一个指向所分配内存空间的起始地址的指针
- 若申请内存空间不成功，则返回**NULL**（值为0）
- 返回值类型：**void ***
 - 通用指针的一个重要用途
 - 将**malloc**的返回值转换到特定指针类型，赋给一个指针

malloc()示例

```
/* 动态分配n个整数类型大小的空间 */  
if ((p = (int *)malloc(n*sizeof(int))) == NULL) {  
    printf("Not able to allocate memory. \n");  
    exit (1);  
}
```

- 调用**malloc**时，用 **sizeof** 计算存储块大小
- 每次动态分配都要检查是否成功，考虑例外情况处理
- 虽然存储块是动态分配的，但它的大小在分配后也是确定的，不要越界使用。

计数动态存储分配函数 `calloc ()`

`void *calloc (unsigned n, unsigned size)`

在内存的动态存储区中分配**n**个连续空间，每一存储空间的长度为**size**，并且分配后还把存储块里全部初始化为**0**

- 若申请成功，则返回一个指向被分配内存空间的起始地址的指针
- 若申请内存空间不成功，则返回**NULL**

- **malloc**对所分配的存储块不做任何事情
- **calloc**对整个区域进行初始化

动态存储释放函数 `free()`

`void free (void *ptr)`

释放由动态存储分配函数申请到的整块内存空间，
`ptr`为指向要释放空间的首地址。

当某个动态分配的存储块不再用时，要及时
将它释放

分配调整函数 `realloc()`

`void *realloc (void *ptr, unsigned size)`

更改以前的存储分配

- **ptr**必须是以以前通过动态存储分配得到的指针
- 参数**size**为现在需要的空间大小
- 如果调整失败，返回**NULL**，同时原来**ptr**指向存储块的内容不变。
- 如果调整成功，返回一片能存放大小为**size**的区块，并保证该块的内容与原块的一致。如果**size**小于原块的大小，则内容为原块前**size**范围内的数据；如果新块更大，则原有数据存在新块的前一部分。
- 如果分配成功，原存储块的内容就可能改变了，因此不允许再通过**ptr**去使用它。

本章小结

■ 指针的概念与定义

- 变量、内存单元与地址的关系
- 指针变量的定义与初始化

■ 指针作为函数参数

- 通过指针参数使函数返回

■ 指针与数组

- 指针、数组与地址的关系
- 数组名作为函数参数

■ 指针与字符串

- 常用字符串处理函数

■ *指针实现内存动态分配

- 能够掌握指针概念，定义指针变量和指针基本运算
- 能够掌握指针作为函数的参数进行熟练编程，通函数调用改变主调函数变量的值
- 作为函数参数进行熟练编程，并能利用指针进行数组相关操作
- 能够使用字符串常用处理函数进行编程，并能使用字符指针进行字符串相关操作
- 了解通过指针实现动态内存分配，并能进行编程



Chap 9 结构

9.1 输出平均分最高的学生信息

9.2 学生成绩排序

9.3 修改学生成绩

本章要点

- 什么是**结构**？**结构与数组**有什么差别？
- 有几种结构的**定义**形式，它们之间有什么不同？
- 什么是结构的**嵌套**？
- 什么是**结构变量**和**结构成员变量**？如何引用结构成员变量？
- 结构变量如何作为**函数参数**使用？
- 什么是**结构数组**？如何定义和使用结构数组？
- 什么是**结构指针**？它如何实现对结构分量的操作？
- 结构指针是如何作为函数的参数的？

9.1 输出平均分最高的学生信息

9.1.1 程序解析

9.1.2 结构的概念与定义

9.1.3 结构的嵌套定义

9.1.4 结构变量的定义和初始化

9.1.5 结构变量的使用

9.1.1 程序解析

例9-1 输出平均分最高的学生信息

- 假设学生的基本信息包括学号、姓名、三门课程成绩以及个人平均成绩。输入 n 个学生的成绩信息，计算并输出平均分最高的学生信息。

运行结果

```
Input n: 3
Input the student's number, name and course scores
No. 1: 101 Zhang 78 87 85
No. 2: 102 Wang 91 88 90
No. 3: 103 Li 75 90 84
num: 102, name: Wang, average: 89.67
```

9.1.1 程序解析

```
# include <stdio.h>
struct student {
    int num;
    char name[10];
    int computer, english, math;
    double average;
};
```

/ 学生信息结构定义 */*
/ 学号 */*
/ 姓名 */*
/ 三门课程成绩 */*
/ 个人平均成绩 */*

```
int main(void)
```

```
{  int i, n;
```

```
    struct student s1, max;      /* 定义结构变量 */
```

```
    printf ("Input n: ");
```

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

```
    printf ("Input the student's number, name and course scores\n");
```

```
    for (i = 1; i <= n; i++){
```

```
        printf ("No.%d: ", i);
```

```
        scanf ("%d%s%d%d%d",&s1.num,s1.name,&s1.math,&s1.english,&s1.computer);
```

```
        s1.average = (s1.math + s1.english + s1.computer) / 3.0;
```

```
        if (i == 1) max = s1;      /* 结构变量 操作 */
```

```
        if (max.average < s1.average)
```

```
            max = s1;
```

```
    }
```

```
    printf("num:%d, name:%s, average:%.2lf\n", max.num, max.name, max.average);
```

```
    return 0;
```

```
}
```

9.1.2 结构的概念与定义

结构与数组比较:

- 都是构造类型，是多个变量的集合
- 数组成员类型相同，结构成员类型不同

- 使用结构来表示学生信息:

```
struct student{  
    int num;           /* 学号 */  
    char name[10];     /* 姓名 */  
    int computer, english, math; /* 三门课程成绩 */  
    double average;    /* 个人平均成绩 */  
};
```

- 结构是C语言中一种新的构造数据类型，它能够把有内在联系的不同类型的数据统一成一个整体，使它们相互关联
- 结构又是变量的集合，可以按照对基本数据类型的操作方法单独使用其变量成员。

9.1.2 结构的概念与定义

- 结构类型定义的一般形式为：

struct 结构名

{

类型名 结构成员名1;

类型名 结构成员名2;

...

类型名 结构成员名n;

};

←

关键字**struct**和它后面的结构名一起组成一个新的数据类型名

结构的定义以分号结束，C语言中把结构的定义看作是一条语句

结构的定义

9.1.2 结构的概念与定义

- 例如，平面坐标结构：

```
struct point  
{  
    float x;  
    float y;  
};
```

•虽然x、y的类型相同，也可以用数组的方式表示，但采用结构进行描述，更贴近事物本质，从而增加了程序的可读性，使程序更易理解

•结构适合用于描述具有多个属性的实体或对象

9.1.3 结构的嵌套定义

- 在我们的实际生活中，一个较大的实体可能由多个成员构成，而这些成员中有些又有可能由一些更小的成员构成。
- 在学生信息中可以再增加一项：“通信地址”，它又可以再划分为：城市、街道、门牌号、邮政编码。

学号	姓名	通信地址				计算机	英语	数学	平均成绩
		城市	街道	门牌号	邮编				

9.1.3 结构的嵌套定义

- 由此，我们可以对其结构类型进行如下重新定义：

struct address {	struct nest_student {
char city[10];	int num;
char street[20];	char name[10];
int code;	struct address addr;
int zip;	int computer, english, math;
};	double average;
	};

• 在定义嵌套的结构类型时，必须先定义成员的结构类型，再定义主结构类型。

9.1.4 结构变量的定义和初始化

■ 在 C 语言中定义结构变量的方式有三种：

1. **单独定义**：先定义一个结构类型，再定义一个具有这种结构类型的变量

```
struct student {  
    int num;           /* 学号 */  
    char name[10];     /* 姓名 */  
    int computer, english, math; /* 三门课程成绩 */  
    double average;    /* 个人平均成绩 */  
};  
struct student s1,s2;
```

9.1.4 结构变量的定义和初始化

2. 混合定义：在定义结构类型的同时定义结构变量

```
struct student {  
    int num;           /* 学号 */  
    char name[10];     /* 姓名 */  
    int computer, english, math; /* 三门课程成绩 */  
    double average;    /* 个人平均成绩 */  
} s1, s2;
```

3. 无类型名定义：在定义结构变量时省略结构名

```
struct {  
    int num;           /* 学号 */  
    char name[10];     /* 姓名 */  
    int computer, english, math; /* 三门课程成绩 */  
    double average;    /* 个人平均成绩 */  
} s1, s2;
```

9.1.4 结构变量的定义和初始化

■ 结构变量的初始化

struct student s1 = {101, "Zhang", 78, 87, 85};

num	name	computer	english	math	average
↓	↓	↓	↓	↓	↓
101	Zhang	78	87	85	

9.1.5 结构变量的使用

1. 结构变量成员的引用

- 在C语言中，使用结构成员操作符“.”来引用结构成员，格式为：

结构变量名 . 结构成员名

```
s1.num = 101;
```

```
strcpy(s1.name, "Zhang");
```

```
nest_s1.addr.zip = 310015;
```


9.1.5 结构变量的使用

2. 结构变量的整体赋值

- 具有相同类型的结构变量可以直接赋值。赋值时，将赋值符号右边结构变量的每一个成员的值都赋给了左边结构变量中相应的成员。

```
struct student s1  
    = {101, "Zhang", 78, 87, 85}, s2;  
s2 = s1;
```

9.1.5 结构变量的使用

特点：可以传递多个数据且参数形式较简单

缺点：对于成员较多的大型结构，参数传递时所进行的结构数据复制使得效率较低

3. 结构变量作为函数参数

- 如果一个C程序的规模较大要以函数的形式进行功能模块的划分和实现
- 如果程序中含有结构数据，则就可能需要用结构变量作为函数的参数或返回值，以在函数间传递数据。
- 例：

```
double count_average ( struct student s );  
main(): s1.average = count_average ( s1 );
```



9.2 学生成绩排序

9.2.1 程序解析

9.2.2 结构数组操作

9.2.1 程序解析

例9-2 输入 n ($n < 50$) 个学生的成绩信息，按照学生的个人平均成绩从高到低输出他们的信息。

```
struct student students[50], temp; /* 定义结构数组 */
```

```
/* 输入 */
```

```
...
```

9.2.1 程序解析

/* 结构数组排序，选择排序法 */

for (i = 0; i < n-1; i++){

index = i;

for (j = i+1; j < n; j++)

if (students[j].average > students[index].average)

/* 比较平均成绩*/

index = j;

temp = students[index];

/* 交换数组元素 */

students[index] = students[i];

students[i] = temp;

}

printf ("num\t name\t average\n"); /* 输出排序后的信息 */

for (i = 0; i < n; i++)

**printf ("%d\t%s\t %.2lf\n", students[i].num,
 students[i].name, students[i].average);**

9.2.2 结构数组操作

- 一个结构变量只能表示一个实体的信息，如果有许多相同类型的实体，就需要使用结构数组。
- 结构数组是结构与数组的结合，与普通数组的不同之处在于每个数组元素都是一个结构类型的变量。

9.2.2 结构数组操作

- 结构数组的定义方法与结构变量类似
struct student students[50];

结构数组**students**，它有**50**个数组元素，
从**students[0]**到**students[49]**，
每个数组元素都是一个结构类型**struct student**
的变量

9.2.2 结构数组操作

■ 结构数组的初始化

```
struct student students[50] = {  
    { 101,"zhang", 76, 85, 78 }, {102, "wang", 83, 92, 86}  
};
```

students[0]	101	Zhang	76	85	78	
students[1]	102	Wang	83	92	86	
...	
students[49]						

9.2.2 结构数组操作

- 结构数组元素的成员引用，其格式为：
结构数组名[下标]. 结构成员名
- 使用方法与同类型的变量完全相同：
`students[i].num = 101;`
`strcpy (students[i].name, "zhang");`
`students[i] = students[k]`

9.3 修改学生成绩

9.3.1 程序解析

9.3.2 结构指针的概念

9.3.3 结构指针作为函数参数

9.3.1 程序解析

例9-3 输入 $n(n < 50)$ 个学生的成绩信息，再输入一个学生的学号、课程以及成绩，在自定义函数中修改该学生指定课程的成绩。

```
int main(void)
{   int course, i, n, num, pos, score;
    struct student students[50];    /* 定义结构数组 */
    ... /* 输入n个学生信息 */
    ... /* 输入待修改学生信息 */
    /*调用函数，修改学生成绩*/
    pos = update_score(students, n, num, course, score);
    ... /*输出修改后的学生信息*/ ...
}
```

9.3.1 程序解析

/* 自定义函数，修改学生成绩 */

```
int update_score (struct student *p, int n, int num, int course, int score)
{
    int i, pos;
    for (i = 0; i < n; i++, p++)      /* 按学号查找 */
        if (p->num == num)
            break;
    if (i < n)      /* 找到，修改成绩 */
    {
        switch (course) {
            case 1: p->math = score; break;
            case 2: p->english = score; break;
            case 3: p->computer = score; break;
        }
        pos = i;      /* 被修改学生在数组中的下标 */
    }
    else      /* 无此学号 */
        pos = -1;
    return pos;
}
```

9.3.2 结构指针的概念

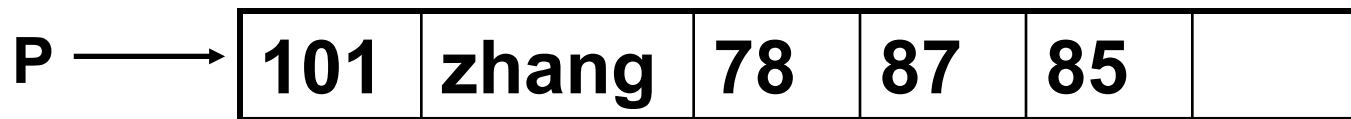
- 指针可以指向任何一种变量，而结构变量也是C语言中的一种合法变量，因此，指针也可以指向结构变量，这就是结构指针。
- 结构指针就是指向结构类型变量的指针

9.3.2 结构指针的概念

struct student s1

= {101, "zhang", 78, 87, 85}, *p;

p = &s1;



9.3.2 结构指针的概念

- 结构指针的使用

- (1) 用*p访问结构成员。如：

(*p).num = 101;

- (2) 用指向运算符“->”访问指针指向的结构成员。如：

p->num = 101;

当p指向结构变量s1时，下面三条语句的效果是一样的：

s1.num = 101;

(*p).num = 101;

p->num = 101;

9.3.3 结构指针作为函数参数

- 结构指针的操作是非常灵活的，如果将结构指针作为函数的参数，可以完成比基本类型指针更为复杂的操作。

■ 例9-3

main():

```
pos = update_score (students, n, num, course, score);
```

自定义函数:

```
int update_score (struct student *p, int n,  
                  int num, int course, int score)
```

函数update_score运行完毕返回主函数后，主函数中的结构数组students中的值已被修改

9.3.3 结构指针作为函数参数

- 与结构变量作为函数参数相比，用结构指针作为函数参数的效率更高。
- 就例 9-3 而言，在函数update_score()中需要修改主函数中结构数组students的数据，根据第 8 章介绍的知识，在此处也只能使用指针作为函数参数的方式才能通过间接访问操作来实现程序功能。

本章总结

■ 结构的概念与定义（含嵌套结构）

■ 结构变量

- 定义
- 初始化
- 使用（成员引用、

- 能够根据实际情况合理定义结构
- 能够使用结构变量与结构数组进行熟练编程

■ 结构数组

- 定义、初始化、结

- 掌握结构指针的操作，并应用于函数参数传递

■ 结构指针

- 概念
- 结构指针操作
- 结构指针作为函数参数



第10章 宏定义+

第11章 函数返回指针

10.3 宏定义与应用（长度单位转换 等）

- 10.3.1 程序解析
- 10.3.2 宏基本定义
- 10.3.3 带参数的宏定义
- 10.3.4 文件包含
- 10.3.5 编译预处理

10.3.1 程序解析

Input mile, foot and inch: 1.2 3 5.1
1.200000 miles=1930.800077 meters
3.000000 feet=91.440000 centimeters
5.100000 inches=12.954000 centimeters

- 例10-5 欧美国家长度使用英制单位，1英里=1609米，1英尺=30.48厘米，1英寸=2.54厘米。请编写程序转换。

```
#include<stdio.h>
#define Mile_to_meter 1609          /* 1英里=1609米 */
#define Foot_to_centimeter 30.48    /* 1英尺=30.48厘米 */
#define Inch_to_centimeter 2.54     /* 1英寸=2.54厘米 */
int main(void)
{
    float foot, inch, mile;          /* 定义英里，英尺，英寸变量 */
    printf("Input mile, foot and inch:");
    scanf("%f%f%f", &mile, &foot, &inch);
    printf("%f miles=%f meters\n", mile, mile * Mile_to_meter);
    /* 计算英里的米数 */
    printf("%f feet=%f centimeters\n", foot, foot *
        Foot_to_centimeter); /* 计算英尺的厘米数 */
    printf("%f inches=%f centimeters\n", inch, inch *
        Inch_to_centimeter); /* 计算英寸的厘米数 */
    return 0;
}
```

10.3.2 宏基本定义

#define 宏名标识符 宏定义字符串

编译时，把程序中所有与宏名相同的字符串，用宏定义字符串替代

#define PI 3.14

#define arr_size 4

说明：

- 宏名一般用大写字母，以与变量名区别
- 宏定义不是C语句，后面不得跟分号
- 宏定义可以嵌套使用

#define PI 3.14

#define S 2*PI*PI

多用于符号常量

10.3.2 宏基本定义

- 宏定义可以写在程序中任何位置，它的**作用范围**从定义书写处到文件尾。
- 可以通过“**# undef**”强制指定宏的结束范围。

宏的作用范围

```
#define A “This is the first macro”
```

```
void f1()  
{  
    printf( “A\n” );  
}
```

```
#define B “This is the second macro”
```

```
void f2( )  
{  
    printf( B );  
}
```

```
#undef B
```

```
int main(void)  
{  
    f1( );  
    f2( );  
    return 0;  
}
```

A 的有效范围

B 的有效范围

10.3.3 带参数的宏定义

例10-6 简单的带参数的宏定义。

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

```
#define MAX(a, b) a > b ? a: b
```

```
#define SQR(x) x * x
```

```
int main(void)
```

```
{
```

```
    int x , y;
```

```
    scanf ("%d%d" , &x, &y) ;
```

```
    x = MAX (x, y);          /* 引用宏定义 */
```

```
    y = SQR(x);              /* 引用宏定义 */
```

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

```
    return 0;
```

```
}
```

10.3.3 带参数的宏定义

例: **#define f(a) (a)*(a)*(a)**

```
int main (void) /* 水仙花数 */
```

```
{ int i, x, y, z;
```

```
  for (i = 1; i < 1000; i++) {
```

```
    x = i%10; y = i/10%10; z = i/100 ;
```

```
    if (f(x)+f(y)+f(z) == i)
```

```
        printf ("%d\n", i);
```

```
  }
```

```
  return 0;
```

```
}
```

各位数字的立方和等于它本身的数。例如153的各位数字的立方和是 $1^3+5^3+3^3=153$

$f(x+y) = (x+y)^3 ? = x+y*x+y*x+y$

带参数的宏

示例 用宏实现两个变量值的交换

```
# define f (a,b,t) t=a; a=b; b=t;
int main ( )
{ int x, y, t ;
    scanf("%d%d" ,&x, &y);
    t=x ; x=y ; y=t ;
    printf("%d %d\n", x, y) ;
    return 0;
}
```

编译时被替换

与函数的区别在哪里？

- 带参数的宏定义不是函数，宏与函数是两种不同的概念
- 宏可以实现简单的函数功能

宏定义应用示例

- 定义宏**LOWCASE**，判断字符**c**是否为小写字母。
#define LOWCASE(c) (((c) >= 'a') && ((c) <= 'z'))
- 定义宏**CTOD**将数字字符（‘0’～‘9’）转换为相应的十进制整数，-1表示出错。
#define CTOD(c) (((c) >= '0') && ((c) <= '9')) ? c - '0' : -1)

练习——带宏定义的程序输出

```
#define F(x) x - 2
```

```
#define D(x) x * F(x)
```

```
int main ( )
```

```
{
```

```
    printf ("%d,%d", D(3), D(D(3))) ;
```

```
    return 0;
```

```
}
```

结果分析

- 阅读带宏定义的程序，先全部替换好，最后再统一计算
- 不可一边替换一边计算，更不可以人为添加括号

$D(3) = x * F(x)$ 先用 x 替换展开

$= x * x - 2$ 进一步对 $F(x)$ 展开，这里不能加括号

$= 3 * 3 - 2 = 7$ 最后把 $x=3$ 代进去计算

$D(D(3)) = D(x * x - 2)$ 先对 $D(3)$ 用 x 替换展开，

$= x * x - 2 * F(x * x - 2)$ 拿展开后的参数对 D 进一步进行宏替换

$= x * x - 2 * x * x - 2 - 2$ 拿展开后的参数对 F 进一步进行宏替换

$= 3 * 3 - 2 * 3 * 3 - 2 - 2 = -13$ 最后把 $x=3$ 代进去计算

运行结果： 7 -13



11.2 字符定位

11.2.1 程序解析

11.2.2 指针作为函数的返回值

11.2.1 程序解析

- 例11-8 输入一个字符串和一个字符，如果该字符在字符串中，就从该字符首次出现的位置开始输出字符串中的字符。例如，输入字符r和字符串program后，输出rogram。
- 要求定义函数match(s, ch)，在字符串s中查找字符ch，如果找到，返回第一次找到的该字符在字符串中的位置（地址）；否则，返回空指针NULL。

11.2.1 程序解析

```
#include <stdio.h>
char *match(char *s, char ch) /* 函数定义 */
{
    while(*s != '\0')
        if(*s == ch) return(s); /* 若找到字符ch */
        else s++;
    return(NULL); /* 没有找到ch */
}

int main(void)
{
    char ch, str[80], *p = NULL;
    printf("Please Input the string:\n");
    scanf("%s", str);
    getchar(); /* 跳过输入字符串和换行符 */
    ch = getchar();
    if((p = match(str, ch)) != NULL) /* 若找到字符ch */
        printf("%s\n", p);
    else printf("Not Found\n");
    return 0;
}
```

运行结果1

Please Input the string:

University v

ersity

运行结果2

Please Input the string:

school a

Not Found

- 函数match()返回一个地址（指针）。
- 在主函数中，用字符指针p接收match()返回的地址，从p指向的存储单元开始，连续输出其中的内容，直至'\0'为止。

函数返回的指针指向可用的存储空间

11.2.1 程序解析

```
#include <stdio.h>
char *match(char ch) /* 函数返回值的类型是字符指针 */
{ char str[30], *s=str;
  scanf("%s", str);
  getchar(); /* 跳过输入字符串和输入字符之间的分隔符 */
  while(*s != '\0')
    if(*s == ch) return(s); /* 若找到字符ch, 返回相应的地址 */
    else s++;
  return(NULL);
}
int main(void )
{ char ch, *p = NULL;
  ch = getchar(); /* 输入一个查找的字符 */
  if((p = match(ch)) != NULL) /* 调用函数match() */
    printf("%s\n", p);
  else printf("Not Found\n");
  return 0;
}
```

思考：有以下运行结果吗？

v University

versity

函数返回的指针指向了不可用的存储空间

11.2.2 指针作为函数的返回值

■ 函数返回值的类型

- 整型
- 浮点型
- 字符型
- 结构类型
- **指针**（返回一个地址）

- 在C语言中，函数返回值也可以是指针，定义和调用这类函数的方法与其他函数是一样的。

Chap 12 文件

12.1 学生成绩文件统计

12.2 用户信息加密和校验

12.3 文件综合应用：资金账户管理

本章要点

- 什么是文件？C文件是如何存储的？
- 什么是文件缓冲系统？工作原理如何？
- 什么是文本文件和二进制文件？
- 怎样打开、关闭文件？
- 怎样编写文件读写程序？
- 怎样编写程序，实现简单的数据处理？

12.1 学生成绩文件统计

【例12-1】有5位学生的计算机等级考试成绩被事先保存在数据文件C:\f12-1.txt(C盘根目录下的文件f12-1.txt，需事先准备好该文件)中，包括学号、姓名和分数，文件内容如下：


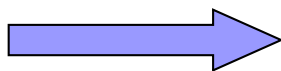
```
301101 Zhangwen 91
301102 Chenhui 85
301103 Wangweidong 76
301104 Zhengwei 69
301105 Guowentao 55
```

请读出文件的所有内容显示到屏幕，并输出平均分。



A screenshot of a Windows Notepad window titled "f12-1.txt - 记事本". The menu bar includes "文件(F)", "编辑(E)", "格式(O)", "查看(V)", and "帮助(H)". The text area contains the following lines:

```
301101 Zhangwen 91
301102 Chenhui 85
301103 Wangweidong 76
301104 Zhengwei 69
301105 Guowentao 55
```



A screenshot of a Windows command prompt window. The title bar shows the path "F:\HOMEVSS\00学校\08项目\00-C教材...". The window displays the following output:

```
301101 Zhangwen 91
301102 Chenhui 85
301103 Wangweidong 76
301104 Zhewei 69
301105 Guowentao 55
Average score: 75
Press any key to continue.
```

例12-1 源程序

```
#include <stdio.h>
#include <stdlib.h>
#include <process.h>
int main(void)
{
    FILE *fp;                                /*1.定义文件指针*/
    long num;  char stname[20];
    int i, score; int avg_score = 0;
    if((fp=fopen("c:\\f12-1.txt","r")) == NULL) /*2.打开文件*/
    {
        printf("File open error!\n");
        exit(0);
    }
}
```

续下页....

例12-1 源程序

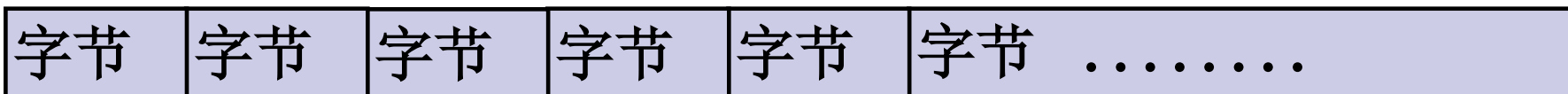
接上页.....

```
/*3.文件处理（逐个读入和处理数据）*/  
for(i=0; i<5; i++)  
{ /*从文件读入成绩保存到变量*/  
    fscanf(fp, "%ld%s%d", &num, stname, &score);  
    avg_score += score; /*统计总分*/  
    /*输出成绩到屏幕*/  
    printf("%ld      %s %d\n", num, stname, score);  
}  
/*输出平均分到屏幕*/  
printf("Average score: %d\n", avg_score/5);  
  
if(fclose(fp)){ /*4.关闭文件*/  
    printf("Can not close the file!\n");  
    exit(0);  
}  
return 0;  
}
```


12.1.2 文件的概念

- **文件**：操作系统中的文件是指驻留在外部介质（如磁盘等）中的一个有序数据集。
- 各种**类型**的文件
 - 程序文件：源文件、目标程序、可执行程序
 - 数据文件（输入/输出）：文本文件、图像文件、声音文件、可执行文件等
- 文件的**特点**：
 - 数据永久保存；数据长度不定；数据按顺序存取

12.1.3 文本文件和二进制文件



C语言中的文件是数据流(由一个个的字节数据组成)
文件的两种数据形式:

- **ASCII码** (文本文件 **text stream**) 字符流
- **二进制码** (二进制文件 **binary stream**) 二进制流

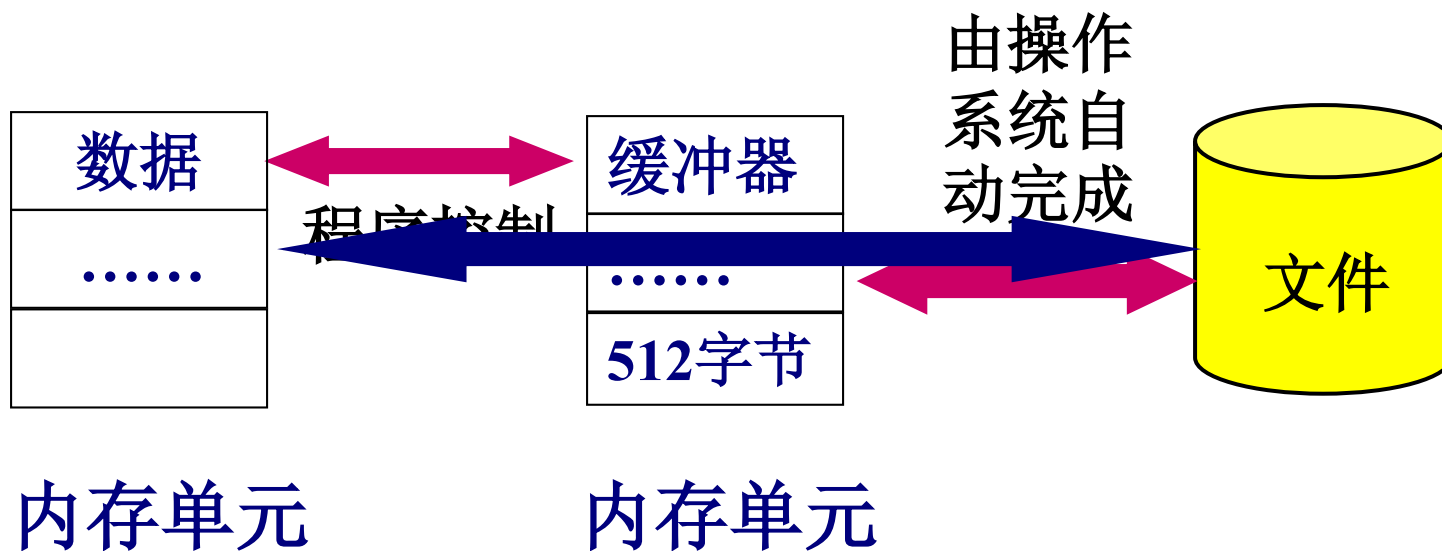
二进制文件是直接把内存数据以二进制形式保存。

例如, 整数**1234**

- 文本文件保存: **49 50 51 52** (4个字符)
- 二进制文件保存: **04D2** (**1234**的二进制数)

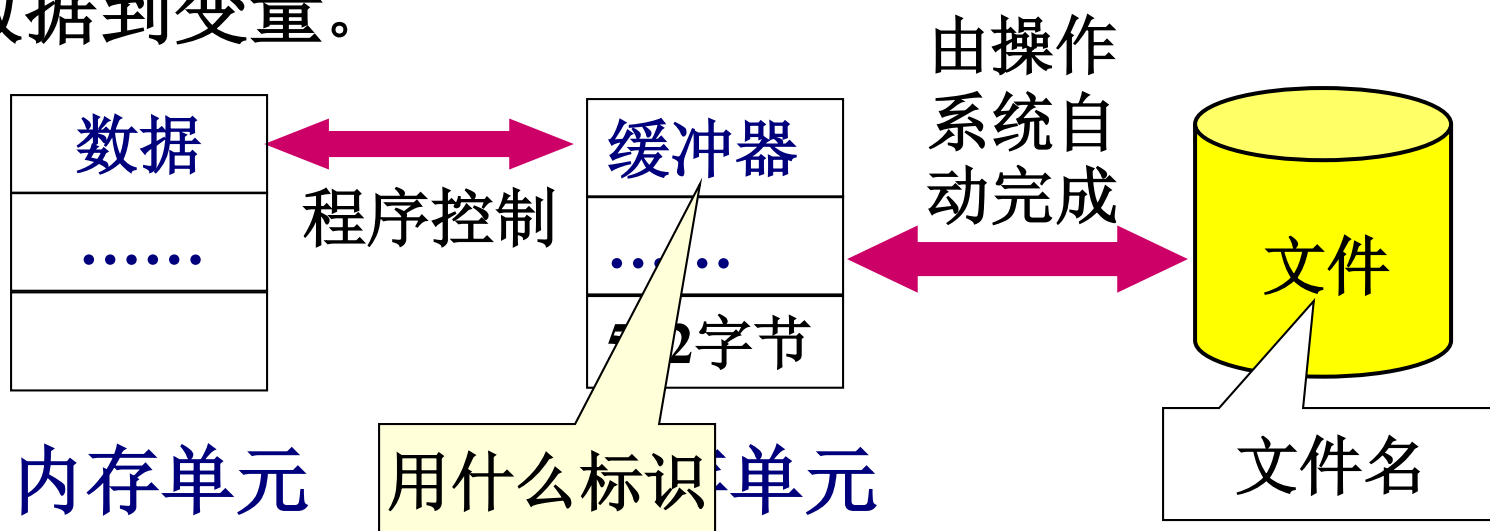
12.1.4 缓冲文件系统

由于磁盘速度慢
直接把数据写到磁盘效率很低



12.1.4 缓冲文件系统

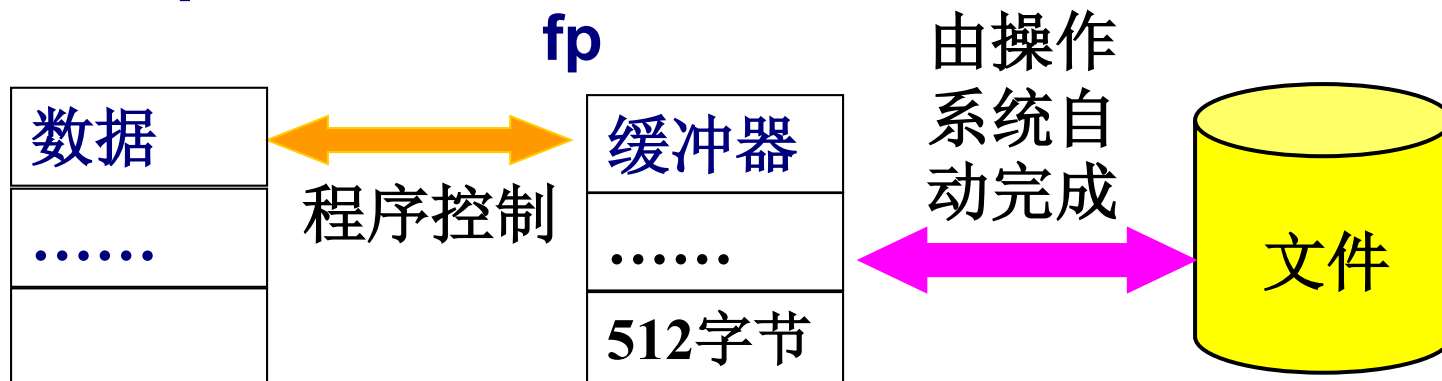
- 向磁盘输出数据：数据 \longrightarrow 缓冲区，装满缓冲区后 \longrightarrow 磁盘文件。
- 从磁盘读入数据：先一次性从磁盘文件将一批数据输入到缓冲区，然后再从缓冲区逐个读入数据到变量。



缓冲文件与文件类型指针

用文件指针指示文件缓冲区中具体读写的位置

FILE *fp;



同时使用多个文件时，每个文件都有缓冲区，用不同的文件指针分别指示。

12.1.5 文件结构与文件类型指针

■ 1. 文件结构与自定义类型typedef

FILE: 结构类型, 用 **typedef** 定义(见stdio.h)

```
typedef struct{  
    short          level;          /* 缓冲区使用量 */  
    unsigned       flags;          /* 文件状态标志 */  
    char           fd;             /* 文件描述符 */  
    short          bsize;          /* 缓冲区大小 */  
    unsigned char  *buffer;        /* 文件缓冲区的首地址 */  
    unsigned char  *curp;          /* 指向文件缓冲区的工作指针 */  
    unsigned char  hold;           /* 其他信息 */  
    unsigned       istemp;  
    short          token;  
} FILE;
```

自定义类型（**typedef**）：

- 将C语言中的已有类型（包括已定义过的自定义类型）重新命名
- 新的名称可以代替已有数据类型
- 常用于简化对复杂数据类型定义的描述

typedef <已有类型名> <新类型名>;

typedef int INTEGER;

int i, j; <====> **INTEGER** i, j;

typedef int* POINT;

int* p1; <====> **POINT** p1;

自定义类型（typedef）的使用方法

- 定义变量
- 变量名 → 新类型名
- 加上 **typedef**
- 用新类型名定义变量

int i

int → **INTEGER**

typedef int **INTEGER**

INTEGER i;

int num[10]

int NUM[10]

typedef int NUM[10]

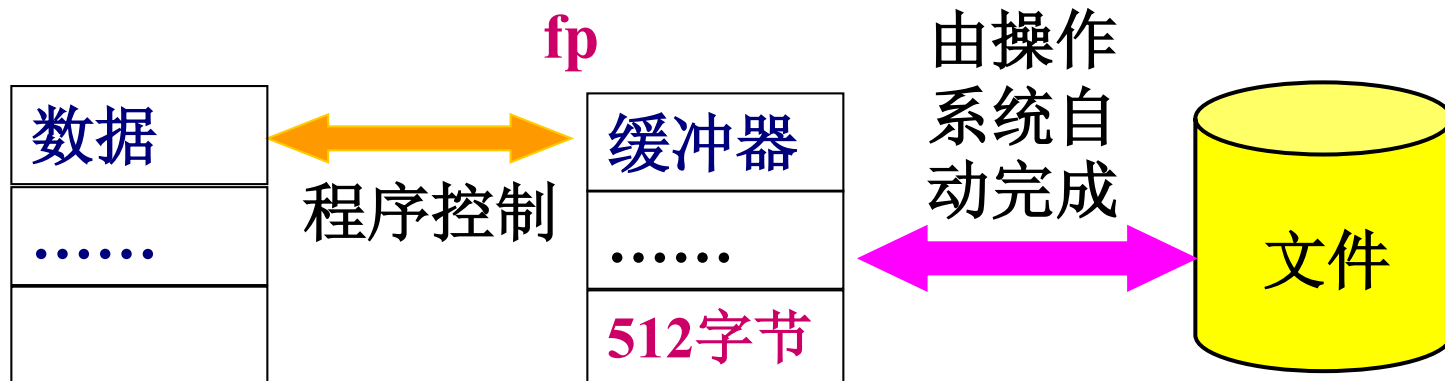
NUM a <===> **int** a[10]

2. 文件类型指针

FILE * fp

如何使**fp**与具体文件挂钩？

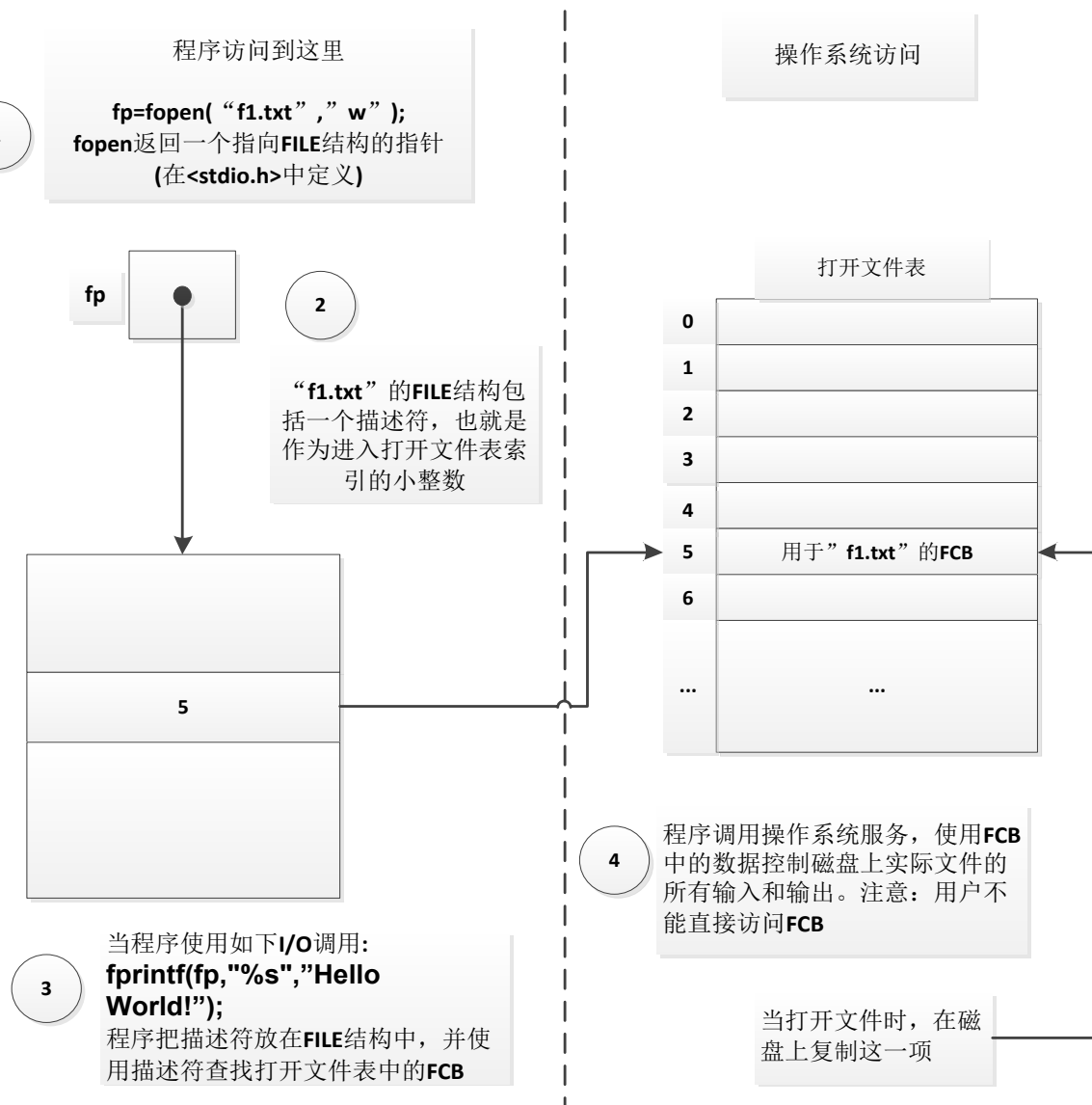
指向文件缓冲区，通过移动指针实现对文件的操作



同时使用多个文件时，每个文件都有缓冲区，用不同的文件指针分别指示。

12.1.6 文件控制块FCB

- 文件控制块**FCB**（**File Control Block**）
- **OS**中对文件的操作控制通过**FCB**，处理的是**FCB**列表
- 一个文件对应一个**FCB**
- 文件缓冲区由程序中**fopen**语句动态创建
- 打开文件时，**FCB**的内容信息被复制到文件缓冲区保存
- 用文件指针指向文件缓冲区实现对文件数据的访问



12.1.7 文件处理步骤

■ 四个步骤:

- ① 定义文件指针
- ② 打开文件: 文件指针指向磁盘文件缓冲区
- ③ 文件处理: 文件读写操作、定位、出错处理
- ④ 关闭文件

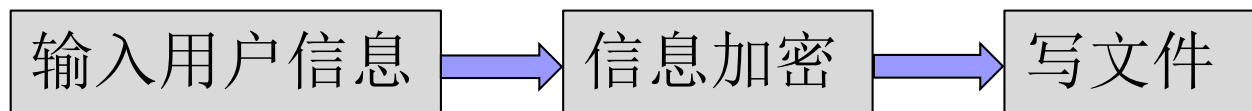
12.2 用户信息加密和校验

【例12-2】为了保障系统安全，通常采取用户帐号和密码登录系统。

系统用户信息存放在一个文件中，系统帐号名和密码由若干字母与数字字符构成，因安全需要文件中的密码不能是明文，必须要经过加密处理。

请编程实现：输入5个用户信息（包含帐号名和密码）并写入文件f12-2.txt。要求文件中每个用户信息占一行，帐号名和加密过的密码之间用一个空格分隔。

密码加密算法：对每个字符ASCII码的低四位求反，高四位保持不变（即将其与15进行异或）。



```
int main(void)
{
    FILE *fp;      /*1.定义文件指针*/
    int i;
    void encrypt(char *pwd);
    struct sysuser su;
    /*2.打开文件，进行写入操作*/
    if((fp=fopen("f12-2.txt","w")) == NULL){
        printf("File open error!\n");  exit(0);
    }
    for(i=1; i<=5; i++){/*3. 将5位用户帐号信息写入文件*/
        printf("Enter %i th sysuser(name password):", i);
        scanf("%s%s", su.username, su.password); /*输入用户名和密码 */
        encrypt(su.password);                      /*进行加密处理*/
        fprintf(fp,"%s %s\n", su.username, su.password); /*写入文件*/
    }
    if(fclosen(fp)){      /*4.关闭文件*/
        printf("Can not close the file!\n");  exit(0);
    }
    return 0;
}
```

12.2.2 打开文件和关闭文件

```
if((fp=fopen("f12-2.txt","w")) == NULL){  
    printf("File open error!\n"); exit(0);  
}
```

fopen("文件名", "文件打开方式")

- 使文件指针与相应文件实体对应起来
- 程序对文件指针进行操作，即**fp**代表磁盘文件

■ 函数**fopen()** 的返回值

- 执行成功，则返回包含文件缓冲区等信息的**FILE**型地址，赋给文件指针**fp**
- 不成功，则返回一个**NULL**（空值）

exit(0): 关闭所有打开的文件，并终止程序的执行

参数**0**表示程序正常结束；非**0**参数通常表示不正常的程序结束

文件打开方式

`fp = fopen("f12-2.txt", "w")`

■ 文件打开方式参数表

文 本 文 件 (ASCII)		二 进 制 文 件(Binary)
使用方式	含 义	
“ r ”	打开只读文件	
“ w ”	建立只写新文件	
“ a ”	打开添加写文件	
“ r+ ”	打开读/写文件	
“ w +”	建立读/写新文件	
“ a +”	打开读/写文件	

文件读写与打开方式

if 读文件

指定的文件必须存在，否则出错；

if 写文件(指定的文件可以存在，也可以不存在)

if 以 "w" 方式写

if 该文件已经存在

原文件将被删去重新建立；

else

按指定的名字新建一个文件；

else if 以 "a" 方式写

if 该文件已经存在

写入的数据将被添加到指定文件原有数据的后面，不会删去原来的内容；

else

按指定的名字新建一个文件（与“w”相同）；

if 文件同时读和写

使用 "r+"、"w+" 或 "a+" 打开文件

关闭文件

```
if( fclose(fp) ){  
    printf( "Can not close the file!\n" );  
    exit(0);  
}
```

fclose(文件指针)

- 把缓冲区中的数据写入磁盘扇区，确保写文件的正常完成
- 释放文件缓冲区单元和**FILE**结构体，使文件指针与具体文件脱钩。

函数**fclose()** 的返回值

- 返回**0**：正常关闭文件
- 返回非**0**：无法正常关闭文件

12.2.3 文件读写

【例12-3】复制用户文件。

将例12-2的用户信息文件**f12-2.txt**文件备份一份，取名为文件**f12-3.txt**。

说明：运行程序前请将文件**f12-2.txt**与源程序放在同一目录下。



文件复制示例

例12-3 源程序

```
#include <stdio.h>
int main(void)
{ FILE *fp1,*fp2;
  char ch;
  if(( fp1 = fopen( "f12-2.txt", "r" )) == NULL){
    printf(" File open error!\n" ); exit(0);
  }
  if(( fp2 = fopen( "f12-3.txt", "w" )) == NULL){
    printf(" File open error!\n" ); exit(0);
  }
  while( !feof( fp1 ) ){                               /*文件结束判断*/
    ch = fgetc( fp1 );                                   /*读文件*/
    if(ch != EOF) fputc(ch, fp2); /*写文件*/
  }
  /*关闭文件*/
  if(fclose(fp1)) { printf("Can not close the file!\n"); exit(0); }
  if(fclose(fp2)) { printf("Can not close the file!\n"); exit(0); }
  return 0;
}
```

打开多个文件

```
if((fp1 = fopen("f12-2.txt", "r")) == NULL){  
    printf("File open error!\n");  
    exit(0);  
}  
if((fp2=fopen("f12-3.txt", "w")) == NULL){  
    printf("File open error!\n");  
    exit(0);  
}
```

C语言允许同时打开多个文件

- 不同的文件对应不同的文件指针
- 不允许同一个文件在关闭前再次打开

文件读写函数

- 字符读写函数: **fgetc()** / **fputc()**
- 字符串读写函数: **fputs()** / **fgets()**
- 格式化读写函数: **fscanf()** / **fprintf()**
- 二进制读写函数: **fread ()** / **fwrite()**
- 其他相关函数:
 - 检测文件结尾函数**feof()**
 - 检测文件读写出错函数**ferror()**
 - 清除末尾标志和出错标志函数**clearerr()**
 - 文件定位的函数**fseek()**、**rewind()**、**ftell()**

见教材附录A, P.348

1. 字符读写函数fgetc和fputc

```
while( !feof(fp1) ){  
    ch = fgetc( fp1 );  
    if(ch!=EOF) fputc(c, fp2);  
}
```

■ 函数fgetc()

ch = fgetc(fp);

从fp所指示的磁盘文件上读
入一个字符到ch

- 区分键盘字符输入函数
getchar()

■ 函数fputc()

□ **fputc(ch, fp);**

把一个字符 **ch** 写到 **fp** 所指
示的磁盘文件上

□ 返回值

- **-1 (EOF):** 写文件失败
- **ch:** 写文件成功

2. 字符串方式读写函数fgets和fputs

■ 函数fputs()

fputs(s, fp);

用来向指定的文本文件写入一个字符串

- **s**: 要写入的字符串，结束符' \0'不写入文件。
- 函数返回值
 - 执行成功，函数返回所写的最后一个字符
 - 否则，函数返回**EOF**

字符串方式读写函数fgets和fputs

■ 函数fgets()

fgets(s, n, fp);

从文本文件中读取字符串

- **s**: 可以是字符数组名或字符指针；**n**: 指定读入的字符个数；**fp**: 文件指针
- 函数被调用时，最多读取**n-1**个字符，并将读入的字符串存入**s**所指向内存地址开始的**n-1**个连续的内存单元中。

当函数读取的字符达到指定的个数，或接收到换行符，或接收到文件结束标志**EOF**时，将在读取的字符后面自动添加一个' \0'字符；若有换行符，则将换行符保留（换行符在' \0'字符之前）；若有**EOF**，则不保留

□ 函数返回值

- 执行成功，返回读取的字符串；
- 如果失败，则返回空指针，这时，**s**的内容不确定

例12-4

例12-2的f12-2.txt文件保存着系统用户信息，编写一个函数**checkUserValid()**用于登录系统时**校验用户**的合法性。检查方法是：

- 在程序运行时输入用户名和密码，然后在用户文件中查找该用户信息，如果用户名和密码在文件中找到，则表示用户合法，返回**1**，否则返回**0**。
- 程序运行时，输入一个用户名和密码，调用**checkUserValid()**函数，如果返回**1**，则提示“**Valid user!**”，否则输出“**Invalid user!**”。

提示：合法性检查的规则。由于文件中的用户名和密码按行存取，把一行看作整体得字符串**s1**，将输入的用户名和密码加密后生成另一个字符串**s2**，然后通过比较**s1**和**s2**，来确定文件中是否存在用户。

例12-4源程序

*/*校验用户信息的合法性，成功返回1，否则返回0*/*

int checkUserValid(**struct** sysuser *psu)

{ **FILE** *fp;

char usr[30], usr1[30], pwd[10]; **int** check=0; */*检查结果变量，初始化为0*/*

*/*连接生成待校验字符串*/*

strcpy(usr, psu->username); */*复制psu->username到usr1 */*

strcpy(pwd, psu->password); */*复制psu->password到pwd */*

encrypt(pwd); */*调用例12-2的encrypt对密码进行加密*/*

*/*连接usr、空格、pwd和\n构成新字符串usr，用于在文件中逐行检查*/*

strcat(usr, " "); strcat(usr,pwd); strcat(usr,"\n");

*/*打开文件"f12-2.txt"读入*/*

if((fp=fopen("f12-2.txt","r")) == **NULL**){

printf("File open error!\n"); exit(0); }

*/*从文件读入用户信息数据，遍历判断是否存在*/*

while(!feof(fp)){

fgets(usr1,30,fp); */*读入一行用户信息作为一个字符串到usr1*/*

if(strcmp(usr, usr1) == 0){ */*比较判断usr与usr1是否相同*/*

check=1; **break**; }

}

if(fclose(fp)){ printf("Can not close the file!\n"); exit(0); } */*关闭文件*/*

return check;

}

回顾例12-2写文件的格式是
fprintf(fp,"%s %s\n",
su.username, su.password);

用户信息合法性校验函数

3. 格式化文件读写fscanf和fprintf

- **fscanf** (文件指针, 格式字符串, 输入表);
 - **fprintf** (文件指针, 格式字符串, 输出表);
- 指定格式的输入输出函数

```
FILE *fp; int n; float x;
```

```
fp = fopen("a.txt", "r");
```

```
fscanf(fp, "%d%f", &n, &x);
```

表示从文件a.txt分别读入整型数到变量n、浮点数到变量x

```
fp = fopen("b.txt", "w");
```

```
fprintf(fp, "%d%f", n, x);
```

表示把变量n和x的数值写入文件b.txt

4. 数据块读写fread()和fwrite()*（不作要求）

■ fread(buffer, size, count, fp);

从二进制文件中读入一个数据块到变量

■ fwrite(buffer, size, count, fp);

向二进制文件中写入一个数据块

□ **buffer**: 指针，表示存放数据的首地址；

□ **size**: 数据块的字节数

□ **count**: 要读写的数据块块数

□ **fp**: 文件指针

12.2.4 其他相关函数

■ 函数feof()

feof(fp) ;

判断**fp**指针是否已经
到文件末尾，

函数返回值

- 1: 到文件结束位置
- 0: 文件未结束

■ 函数rewind()

rewind(FILE *fp);

定位文件指针，使文件指针指向读写文件的首地址，即打开文件时文件指针所指向的位置。

其他相关函数

■ 函数fseek()——用来控制指针移动

fseek(fp, offset, from);

□ **offset:** 移动偏移量, long型

□ **from:** 起始位置, 文件首部、当前位置和文件尾部分别对应0,1,2, 或常量**SEEK_SET**、**SEEK_CUR**、**SEEK_END**。

例如:

fseek(fp, 20L, 0): 将文件位置指针移动到离文件首20字节处

fseek(fp, -20L, SEEK_END): 将文件位置指针移动到离文件尾部前20字节处

■ 函数ftell()

ftell(文件指针);

获取当前文件指针的位置, 即相对于文件开头的位移量 (字节数)

□ 函数出错时, 返回-1L

其他相关函数

- **ferror()**函数：函数用来检查文件在用各种输入输出函数进行读写是否出错，若返回值为**0**，表示未出错，否则表示有错

调用形式为：**ferror(文件指针);**

- 文件指针必须是已经定义过的

- 函数**clearerr()**

clearerr(文件指针);

用来清除出错标志和文件结束标志，使它们为**0**

12.3 文件综合应用：个人资金账户管理

■ 12.3.1 顺序文件和随机文件

按照C程序对文件访问的特点来分，文件可分为顺序访问文件和随机访问文件，简称为顺序文件和随机文件。

■ 逻辑随机访问

```
int GetRecord(FILE *fp, struct RECORD *r, int n);  
  
{ . . .  
  
    fseek(fp, (long)sizeof(struct RECORD) *(n-1), SEEK_SET);  
  
    fscanf(.....); . . . . }  
  
}
```

■ 物理随机访问

由存储介质的特性决定：磁带机是顺序存取设备，硬盘是随机~~

12.3.2 个人资金帐户的管理

■ 要求

- 个人资金账户的信息统一放在随机文件中, 该随机文件包括的数据项有记录ID、发生日期、发生事件、发生金额（正+的表示收入, 负-表示支出）和余额。每记录一次收支, 文件要增加一条记录, 并计算一次余额。
- 程序实现3个功能, 包括: 1) 可以创建该文件并添加新收入或支出信息; 2) 可以显示所有记录列表, 得知资金账户的收支流水帐; 3) 查询最后一条记录, 获知账户最后的余额。

cashbox.dat文件的部分内容

LogID	CreateDate	Note	Charge	Balance
1	2006-06-01	alimony	500.00	500.00
2	2006-06-08	shopping	-300.00	200.00
3	2006-06-15	shopping	-60.00	140.00
4	2006-06-20	workingpay	200.00	340.00
5	2006-08-01	scholarship	1000.00	1340.00

.....

本章总结

■ 文件的概念

- 文本文件和**二进制文件***
- 文件缓冲系统
- 文件结构，文件指针，自定义类型

■ 文件的打开与关闭

- 文件处理实现过程

■ 文件读写操作与常用文件操作函数

■ 文件综合应用

- 1) 掌握文件的概念，文件缓冲系统的基本原理，文件读写操作实现的基本过程；
- 2) 能熟练使用文件进行编程。
- 3) 掌握常用的文本文件读写操作函数。
- 4) 了解顺序文件和随机文件的应用。