

# 课程简介及教学安排

## 一、教学目的

- 掌握计算机程序设计的基本知识和程序设计的一般方法，了解进行科学计算的一般思路。
- 掌握C语言的基本内容及编程技巧，培养学生具备应用计算机解决和处理问题的思维方法与基本能力，为进一步学习和应用计算机打下基础。

# 课程简介及教学安排

## 二、教学安排及学时分配

1. 总学时：46学时（2~15周）

授课学时：26学时（13次）

上机：16学时（8次）

2. 学习方式

教师授课+上机实践+学生自学

# 课程简介及教学安排

## 三、课程特点

1. 语言丰富、表达力强、功能强大
2. 语言简洁、紧凑,使用方便、灵活
3. 概念复杂、规则繁多、内容枯燥
4. 语法限制不太严格, 程序设计自由度大, 容易出错
5. 实践操作性很强, 不仅要掌握理论知识, 实验不容忽视
6. 上机实验不仅可消化和巩固知识, 也可验证学习中的疑难问题, 激发和提高学习C程序设计的兴趣

# 课程简介及教学安排

## 四、要求

- 1.坚持出勤、认真听课，适当做笔记；课后看书、适当记忆。
- 2.认真进行上机编程训练，上机不允许做与课程无关的事情，按时、独立完成上机作业，不得抄袭、拷贝作业。
- 3.有问题及时问，多与老师沟通。

# 课程简介及教学安排

## 五、教材及参考书

- ① C程序设计. 谭浩强编著. 清华大学出版社.
- ② C程序设计题解与上机指导. 谭浩强编著. 清华大学出版社.
- ③ 从问题到程序—程序设计与C语言引论. 裘宗燕编著. 机械工业出版社,2005.9
- ④ Balagurusamy. Programming in ANSI C.清华大学出版社

## 六、考试方式

期末笔试

## 七、成绩计算

笔试成绩（80%）+上机成绩（20%）

# 课程简介及教学安排

## 八、教师联系方式与作业要求

教师	张云飞 教授	祝明 讲师	窦田天 硕士生	
办公室	新主楼 C-1116		新主楼 D-512	
电话	82339399		82313167	
手机	13581908611		15811387409	
e-mail	cloud_zhang@ buaa.edu.cn		doutt@163.com	

# 作业要求

- 在上机实验课后，完成上机作业，将其编辑成电子文档，于当天晚上12点以前发送给助教（如果迟交，上机成绩将降低一等）。

- 例如格式为：1-100511018

4-1. ....

4-2 . ....

- 提交内容主要是：源程序，输入参数、运算结果的屏幕截图。

## 九、课程安排

周序	章	内容	计算机实践
2	1, 2	C语言概述与算法	
3	3	数据类型、运算符与表达式	
4	3	顺序程序设计与习题课	1
5	4	选择结构程序设计	2
6		清明节	
7	5	循环结构程序设计	3
8	6	数组（1）	4
9	6	数组（2）与习题课	5
10	7	函数（1）	6
11	7	函数（2）	7
12		预处理命令与习题课	8
13	8	指针（1）	9
14	8	指针（2）	
15	10	文件	



# 张云飞简介

- 博士，教授，硕士生导师

## 1) 科研工作

- 1988年开始从事飞行器设计专业的教学与科研工作
- 曾参与某型无人机设计、制造、试飞工作
- 在国内核心刊物上发表论文三十余篇，合写专著和手册3部
- 获中航总科技进步二等奖2次，三等奖1次，“航空预研三等功”1次和“航空科技之星”奖章1枚，航空基金二等奖1次

## 2) 教学工作

- 主讲本科生《C语言程序设计》、研究生《飞行器总体效能分析》
- 带出20多名硕士研究生和工程硕士
- 曾任本科1988级、1993级、2001级、2005级班主任，多次评为优秀班主任

## 3) 主要研究方向

- 飞机总体设计技术
- 飞行器隐身技术
- 飞机飞行性能计算
- 飞行器突防仿真与隐身效能评估等



# 第一章 C语言概述

# 1.0程序和程序语言

- ❖ 程序：在生活上指完成某项事务的一套既定活动方式或者活动过程。
- ❖ 学生正常上课一天的行为：

1. 起床
2. 洗漱
3. 早饭
4. 上课
5. 午饭
6. 上图书馆
7. 晚饭

分解：

上图书馆

- 查图书卡片
- 上网搜索图书
- 进入图书室找图书
- 办理借书手续

# 1.0程序和程序语言

## ❖ 计算机、程序与程序设计

### 🌿 计算机：

- ✓ 机器，计算，执行指令
- ✓ 本质特征：按程序（指令集）工作，称为执行程序
- ✓ 通用机器：采用现代生产方式大量生产
- ✓ 专用机器：处理专门程序，完成某种特殊工作的专门机器

### 🌿 程序设计：人们编制计算机程序的工作称为程序设计或者编程

# 1.0 程序和程序语言

## ❖ 程序设计语言及其发展

❧ 自然语言：人为对象

❧ 程序设计语言：计算机为对象，又称编程语言、程序语言

❖ 机器语言：二进制形式  $a \times b + c$

00000001000000001000 —— 将单元1000的数据装入寄存器0

00000001000100001010 —— 将单元1010的数据装入寄存器1

00000101000000000001 —— 将寄存器1的数据乘到寄存器0的原有数据上

00000001000100001100 —— 将单元1100的数据装入寄存器1

00000100000000000001 —— 将寄存器1的数据加到寄存器0的原有数据上

00000010000000001110 —— 将寄存器0的数据存入单元1110

指令      目的      源

# 1.0程序和程序语言

- ❖ 汇编语言：用汇编系统翻译成二进制形式
- ❖ 高级程序语言

load 0 a ——将单元a的数据装入寄存器0

load 1 b ——将单元b的数据装入寄存器1

mult 0 1 ——将寄存器1的数据乘到寄存器0的原有数据上

load 1 c ——将单元c的数据装入寄存器1

add 0 1 ——将寄存器1的数据加到寄存器0的原有数据上

save 0 d ——将寄存器0的数据存入单元d

# 1.0程序和程序语言

## ❖ 高级语言及其实现

### 🌀 高级程序语言：

- ✓ 接近人习惯的描述形式，容易使用
- ✓ 开发出更多的应用系统
- ✓  $d=a*b+c$

### 🌀 高级语言的基本实现方式：编译和解释

- ❖ 编译方式：高级程序语言书写的程序 $\Rightarrow$ 翻译软件 $\Rightarrow$ 计算机执行
- ❖ 解释方式：高级程序语言书写的程序 $\Rightarrow$ 解释软件 $\Rightarrow$ 计算机执行

# 1.0程序和程序语言

✧ 高级程序语言：

- ✓ FORTRAN, FORTRAN 90(FORMular TRANslation)
- ✓ C, C++
- ✓ PASCAL, Ada, Java
- ✓ 非常规: LISP, Smalltalk, PROLOG, ML



# 1.1 C语言出现的历史背景(1)

- ❖ C语言广泛流行，写系统软件，写应用软件
- ❖ 为了编写计算机操作系统：
  - ❖ 1) 采用汇编语言，可以直接对硬件进行操作，例如可以直接访问内存，效率高；但其依赖于计算机硬件，可读性和移植性都比较差。
  - ❖ 2) 采用高级语言，与汇编语言相反。
- ❖ 1960年的ALGOL60 是一种面向问题的高级语言，离硬件较远，不宜用于编写操作系统。
- ❖ 1963年剑桥大学的CPL(combined program language) 接近硬件，但规模大，难以实现。

# 1.1 C语言出现的历史背景(2)

- ❖ 1967年剑桥的 Martin Richard推出 BCPL
- ❖ 1970年美国电报电话公司(AT&T)贝尔实验室的Ken Thompson以BCPL为基础，进一步简化，设计出很简单且接近硬件的B语言，并用B语言写了第一个UNIX操作系统。
- ❖ 1972~1973年Dennis M. Ritchie在 B语言的基础上设计出C语言，保持了BCPL 和B语言精炼、接近硬件等优点，克服了简单、数据无类型等缺点。
- ❖ 1973年， Ken Thompson与Dennis.M.Ritchie把他们1969年用汇编语言编写的UNIX的90%以上用C语言改写为UNIX 5。

# 1.1 C语言出现的历史背景(3)

- ❖ 一九七八年由美国电话电报公司(AT&T)贝尔实验室正式发表了C语言，同时**B. W.Kernighan**和**D.M.Ritchie** (简称K&R)1978年发表名著《The C Programming Language》，成为各种C语言版本的基础。C开始进入其它操作系统，很快在各种大、中、小、微型机上得到广泛应用。
- ❖ 1987年美国国家标准化协会ANSI在K&R的著作基础上公布了新标准称 87 ANSI C。
- ❖ 1990国际标准化组织将其命名为ISO C 的标准 ISO9899-1990。

# 1.2 C语言的特点（1）

C语言的主要特点如下：

1. 是一种较简单的语言，容易入门，上手快。
2. 语言精练，层次清晰，便于按模块化方式组织程序，易于调试和维护。

`i+=2;`    `i=i+2`

`if (e) s ;` 如果 **e** 为真，则执行 **s**；否则执行下一条语句

`for (s1; s2; s3) s4;`

1) 执行表达式**s1**

2) 如果表达式**s2**为真，则执行 **s4**；否则，跳出本循环

3) 执行表达式**s3**，然后返回2)

## 1.2 C语言的特点（2）

3. 具有丰富的运算符和数据类型，语言表达能力强，具有较强的数据类型构造能力。
4. 可移植性强，广泛移植到了各类计算机上，形成多种版本的C语言。
5. 生成代码质量高，程序执行效率高。只比汇编程序的目标代码效率低10%~20%。可以直接访问内存的物理地址，进行位(bit)一级的操作。
6. 由于实现了对硬件的编程操作，因此C语言集高级语言和低级语言的功能于一体，既可用于系统软件的开发，也适合于应用软件的开发。
7. C语言的限制少，对编程人员要求高。

## 1.2 C语言的特点（3）

### ❖ C语言版本

目前最流行的C语言有以下几种：

- 🌿 Microsoft C 及后续 Visual C/C++
- 🌿 Borland公司的 Turbo C及后续 Borland C/C++
- 🌿 Watcom C/C++ 和Symantec C/C++

这些C语言版本不仅实现了**ANSI C**标准，而且在此基础上各自作了一些扩充，使之更加方便、完美。



## 1.2 C语言的特点（4）

### ❖ 面向对象的程序设计语言

- ❧ 在C的基础上，一九八三年又由贝尔实验室的Bjarne Stroustrup推出了C++。C++进一步扩充和完善了C语言，成为一种面向对象的程序设计语言。
- ❧ C++目前流行的最新版本是Borland C++, Symantec C++和Microsoft Visual C++。
- ❧ C++提出了一些更为深入的概念，它所支持的这些面向对象的概念容易将问题空间直接地映射到程序空间，为程序员提供了一种全新的思维方式和编程方法，因而也增加了整个语言的复杂性，掌握起来有一定难度。

## 1.3 简单的C程序介绍

### 例1.1

```
main( )  
{  
    printf("Hello, everyone! %c\n",2);  
}
```

程序的运行结果:

Hello, everyone! ☺

**main**是主函数的函数名，表示这是一个主函数。每一个C源程序都必须有，且只能有一个主函数(**main**函数)。

函数调用语句**printf**函数的功能是把要输出的内容送到显示器去显示。**printf**函数是一个由系统定义的标准函数，可在程序中直接调用。



# 1.3 简单的C程序介绍

## 例1.2

```
#include "stdio.h" //称为文件包含命令
#include "math.h" //扩展名为.h的文件称为头文件或首部文件
main()
{
    double x,s; //定义两个实数变量，以被后面程序使用
    printf("input number:\n"); //显示提示信息
    scanf("%lf",&x); //从键盘获得一个实数x
    s=sin(x); //求x的正弦，并把它赋给变量s
    printf("sine of %lf is %lf\n",x,s); //显示程序运算结果
} //main函数结束
```

程序的运行结果:

Sin of 1.5708 is 0.9999999



# 1.3 简单的C程序介绍

例1.3:从键盘输入两个整数,并将最大的数显示出来。

```
#include "stdio.h" /*包含预处理语句*/
```

```
void main( )//主函数
```

```
{ int a,b,c;
```

```
printf("Please input two integers:\n");
```

```
scanf("%d,%d", &a, &b);
```

```
c=max(a,b);
```

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

```
}
```

```
int max(int x,int y)//子函数
```

```
{ int z;
```

```
if(x>y) z=x;
```

```
else z=y;
```

```
return(z );
```

```
}
```

程序运行情况:

**Please input two integers:**

8,5 ✓

max=8



## 1.3 简单的C程序介绍

❖ **例1.4:**一个百万富翁碰到一个陌生人，陌生人跟他谈了一个换钱的计划。

- ✎ 陌生人说：我每天给你10万元，而你第一天给我一元；第二天我仍给你十万，你给我二元；第三天我仍给你十万，你给我四元；……。你每天给我的钱是前一天的两倍，直到满30天。百万富翁非常高兴，欣然接受了这个契约。
- ✎ 请编写一个程序，计算这 $n$  ( $0 \leq n \leq 30$ )天中，陌生人给了富翁多少钱，富翁给了陌生人多少钱。

```
#include<stdio.h>
#include<math.h>
❖ void main()
❖ {
❖     int i,day;
❖     double stranger_money=1,stranger_money_daily=1;
❖     double millionaire_money=100000,money=100000;

❖     printf("Which day you want to output? ");
❖     scanf("%d",&day);
❖     for(i=1;i<=day;i++)
❖     {
❖         printf("The %d th day:\n",i);
❖         printf("millionaire's money=%12.1f\n",millionaire_money);
❖         printf("  stranger's money=%12.1f\n\n",stranger_money);
❖         stranger_money_daily=2*stranger_money_daily;
❖         stranger_money=stranger_money+stranger_money_daily;
❖         millionaire_money=millionaire_money+money;
❖     }
❖ }
```

通过以上例子可以看出：

## 1. 一个C语言源程序：

- 可以由一个或多个源文件组成
- 每个源文件由一个或多个函数组成( C程序是由函数构成的)
- 其中必须有且仅有一个主函数main( )

函数容易实现程序的模块化。 举例

(C中不存在主程序、子程序的概念，C的结构是函数和语句)

## 2. 源程序中可以有预处理命令(include 命令仅为其中的一种)，预处理命令通常应放在源文件或源程序的最前面。 举例

```
#include "stdio.h"      /*包含预处理语句*/  
int max(int x,int y);//函数原型
```

```
void main( )//主函数  
{ int .....  
  .....  
}
```

```
int max(int x,int y)//子函数  
{ int .....  
  .....  
}
```

3. 一个可执行的**C**语言程序总是从**main**函数开始执行，而不论其在整个程序中的位置如何。 [举例](#)

4. 每一个说明，每一个语句都必须以分号结尾。 [举例](#)



```
#include "stdio.h" /*包含预处理语句*/
```

```
int max(int x,int y)//子函数
```

```
{ int z;
```

```
  if(x>y) z=x;
```

```
  else z=y;
```

```
  return(z );
```

```
}
```

```
void main( )//主函数
```

```
{ int a,b,c;
```

```
  printf("Please input two integers:\n") ;
```

```
  scanf("%d,%d" , &a, &b);
```

```
  c=max(a,b);
```

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

```
}
```

程序开始

程序结束



**说明：**在以下三种情况下不允许有分号：

错误

正确

**a.**预处理语句后面不使用分号；

```
#include "stdio.h" ;
```

```
#include "stdio.h"
```

**b.**函数头后面不使用分号；

```
sum( ) ;  
{ ..... }
```

```
sum( )  
{ ..... }
```

**c.**右花括号“}”后面不使用分号。

```
for(i=1;i<=day;i++)  
{  
    .....  
} ;
```

```
for(i=1;i<=day;i++)  
{  
    .....  
}
```

5. 标识符，关键字之间必须至少加一个空格以示间隔。若已有明显的间隔符，也可不再加空格来间隔。 [举例](#)

6. C对输入输出实行函数化。 [举例](#)

7. C程序书写格式自由，一行内可以写几个语句，一个语句也可以分写在多行上。

```
void main()
{
    int i,day;
    double stranger_money,stranger_money_daily;
    stranger_money=1;stranger_money_daily=1;
    printf("Which day you want to output? ");
    scanf("%d",&day);
    for(i=1;i<=day;i++)
    {
        .....
        printf("millionaire's money=%12.1f\n",
millionaire_money);
        printf("  stranger's money=%12.1f\n\n",
stranger_money);
        .....
    }
}
```

8. 可用`/*.....*/`对C程序中的任何部分作注释，注释可以写在程序的任何位置上，“`/*`”与“`*/`”也可不在同一行上。[举例](#)

```
#include "stdio.h" /*预处理语句，用于将标准的输入输出函数头文件包含到本函数中*/
```

```
#include "math.h" /*预处理语句，包含数学函数库*/
```

```
void main( )//主函数
```

```
{
```

```
.....
```

```
}
```

9. 在C语言中，大小写字母是有区别的。

- C语言习惯用小写字母
- 除了变量名（包括变量和字符常量）和字符、字符串以外，**C语言中的所有关键字、库函数名均采用小写字母**

```
#include "stdio.h"
#include "math.h"
void main( )//主函数
{ int a,b,c;
  double A;
  c=max(a,b);
  A=a*b*sin(c);
}
```

## 1.3 简单的C程序介绍

10. 一个函数由两部分组成：

1) 函数的首部（又称函数头），包括函数名、函数类型、函数参数（形式参数）名、参数类型。

int            max   (int            x,            int            y)

函数类型   函数名   参数类型   参数名   参数类型   参数名

- ❖ 函数名后必须跟一对圆括弧，就像f( ), g( )
- ❖ 函数没有类型时，函数值缺省为整型，例如main(); 函数参数也可以没有。void main()为无类型、无参数的主函数。

## 1.3 简单的C程序介绍

2) 函数体，即函数下面的大括号内的部分，例如

```
main() int max(int x, int y)
```

```
{.....} {.....}
```

函数体一般包括：

声明部分： 如int a,b,c; int a,b,sum;

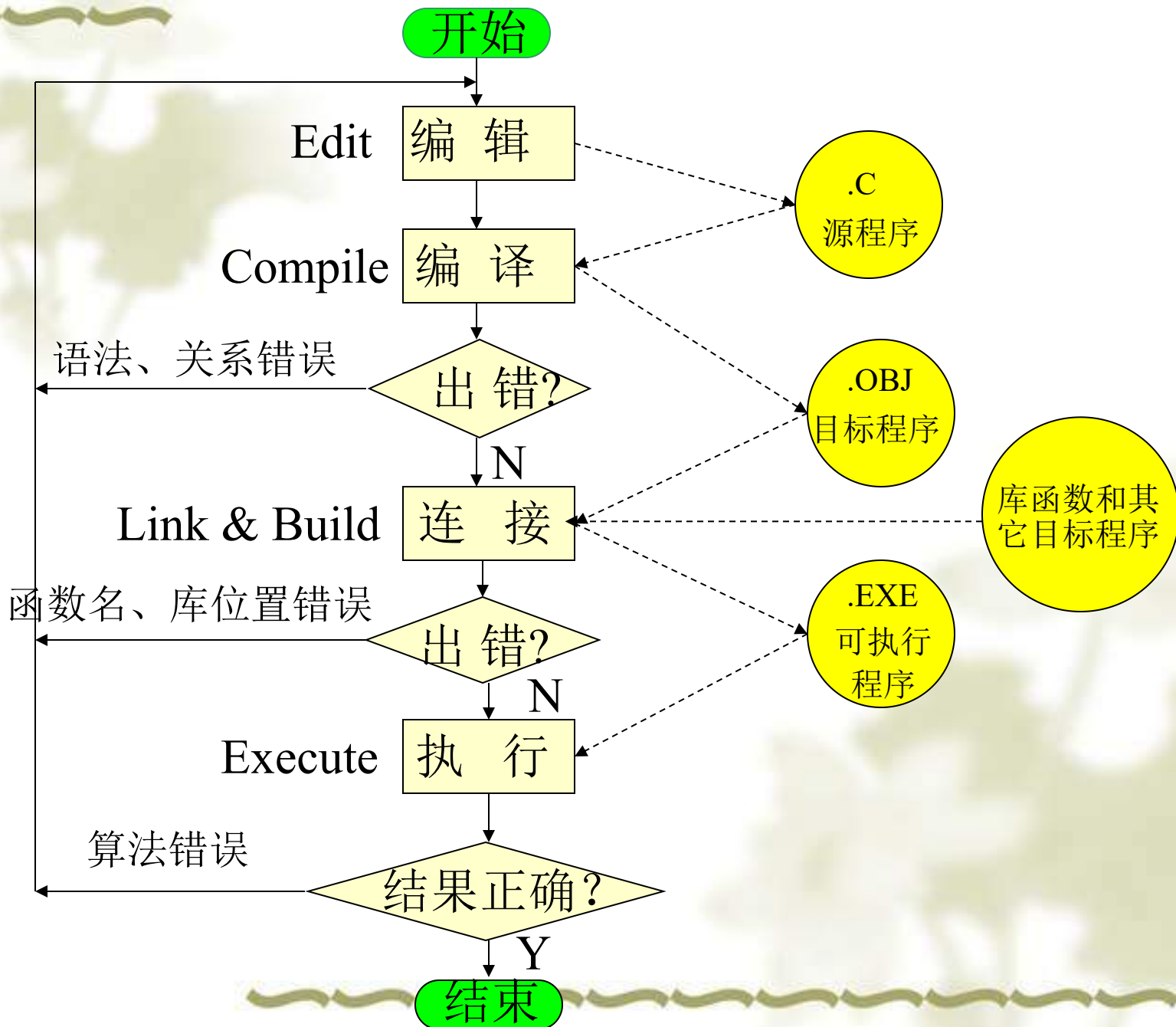
执行部分： 由若干个语句组成



## 1.4 C程序的上机步骤

- 1.编辑：选择适当的编辑程序，将C语言源程序通过键盘输入到计算机中，并以文件的形式存入到磁盘中（.c或者.cpp）
- 2.编译：即将源程序翻译成机器语言程序的过程。编译出来的程序称为目标程序（.obj）
- 3.连接：编译后生成的目标文件经过连接后生成最终的可执行程序（.exe）





```
main( )
```

```
{
```

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

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

```
printf("  Very good!\n");
```

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

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

```
}
```

运行 结果:

\*\*\*\*\*

Very good!

\*\*\*\*\*

```
main( )  
{ int a,b,c,max;  
    printf("input number a,b,c: \n");  
    scanf("%d,%d,%d",&a,&b,&c);  
    max=a;  
    if(max<b) max=b;  
    if(max<c) max=c;  
    printf("MAX: %d",max);  
}
```

运行结果:

input number a,b,c:  
6, 5, 1✓  
MAX: 6

## 第二章 算法简介及程序的基本结构

本章要点：

- ❖ 了解算法的基本概念
- ❖ 掌握程序的基本结构

程序=数据结构+算法+程序设计方法+编程语言

## 2.1 算法的概念

对特定问题求解步骤的一种描述 (algorithm)

计算机算法：

数值算法，有模型，比较成熟，数学程序库

非数值算法，种类繁多，难以规范

## 2.2 简单算法举例 (用自然语言描述法)

例1. 有50个学生，要求将他们之中成绩在80分以上的学号和成绩打印出来。

用 $n$ 表示学号， $n_1$ 代表第一个学生学号， $n_i$ 代表第 $i$ 个学生学号。用 $g$ 代表学生成绩， $g_i$ 代表第 $i$ 个学生成绩，算法可表示如下：

**S1:**  $1 \Rightarrow i$

**S2:** 如果 $g_i \geq 80$ ，则打印  $n_i$  和 $g_i$ ，否则不打印

**S3:**  $i+1 \Rightarrow i$

**S4:** 如果  $i \leq 50$ ，返回S2，继续执行；否则，算法结束。

## 例2 将学生百分成绩按分数段分级的程序。

该算法的核心部分是对输入的每一个数进行比较判断，以确定所属的级别。算法描述如下：

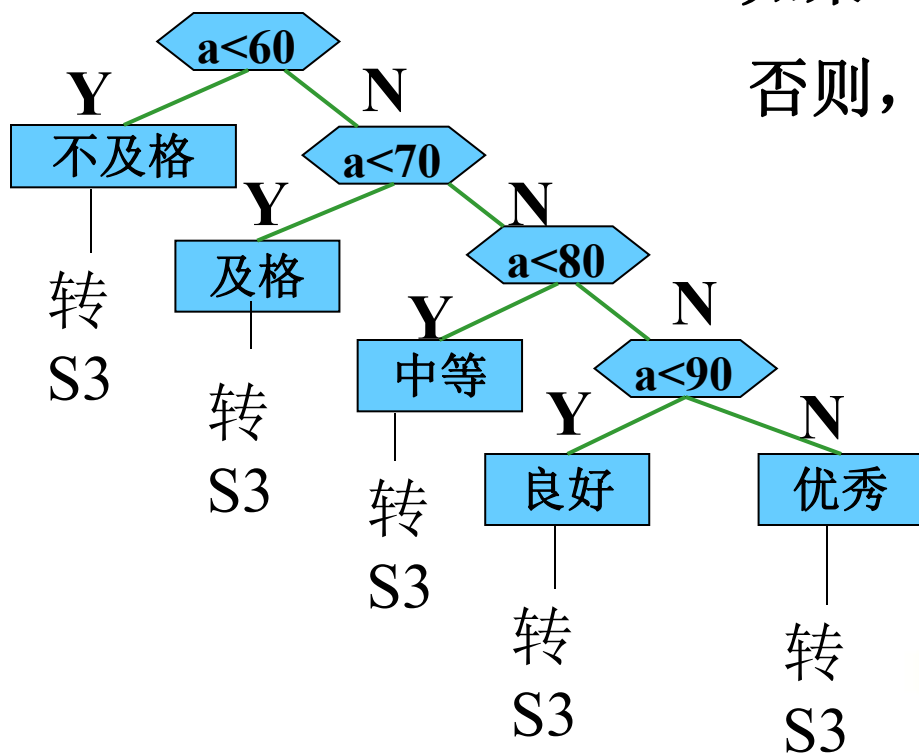
**S1:**  $1 \Rightarrow i$

**S3:**  $i+1 \Rightarrow i$

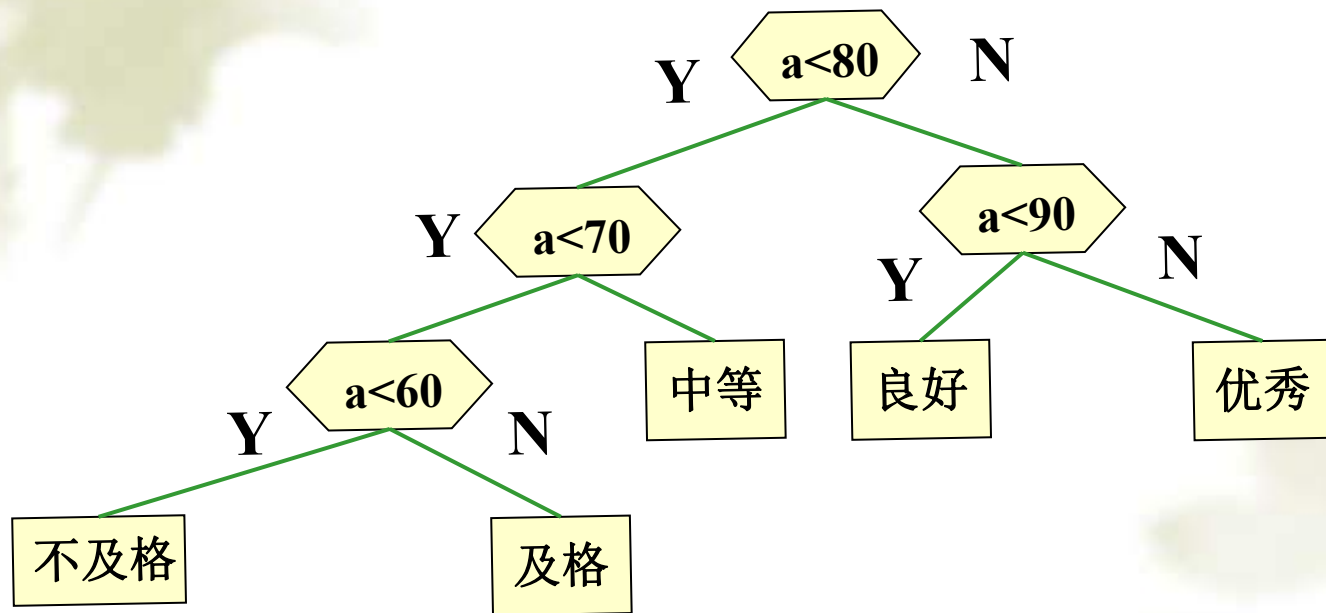
**S2:**

**S4:** 如果  $i \leq \text{num}$ , 返回S2;

否则，算法结束。



第二种算法如下：



分数	0—59	60—69	70—79	80—89	90—99
比例	0.05	0.15	0.4	0.3	0.10



**例3：** 对一个大于或等于3的正整数，判断它是不是一个素数。

方法：将  $n$  ( $n \geq 3$ ) 作为被除数，将2 到  $(n-1)$  各个整数轮流作为除数，如果都不能被整除，则 $n$ 为素数。

算法表示如下：

S1: 输入 $n$ 的值；

S2:  $2 \Rightarrow i$  ( $i$  作为除数) ；

S3:  $n$  被  $i$  除，得余数  $r$ ；

S4: 如果  $r$  等于 0 ， 表示  $n$  能被  $i$  整除，则打印  $n$  “不是素数”， 算法结束； 否则执行S5；

S5:  $i+1 \Rightarrow i$  ；

S6: 如果  $i \leq n-1$ , 返回S3； 否则打印  $n$  “是素数”， 算法结束。

例4：求  $1 - 1/2 + 1/3 - 1/4 + \dots + 1/99 - 1/100$ 。

S1:  $1 \Rightarrow \text{sign}$

//设置符号变量第1项为+

S2:  $1 \Rightarrow \text{sum}$

//设置和变量等于第1项

S3:  $2 \Rightarrow \text{deno}$

//设置分母变量

S4:  $(-1) * \text{sign} \Rightarrow \text{sign}$

//i+1项的符号= $(-1) * i$ 项符号

S5:  $\text{sign} * (1/\text{deno}) \Rightarrow \text{term}$

//计算级数第i+1项的值

S6:  $\text{sum} + \text{term} \Rightarrow \text{sum}$

//累加

S7:  $\text{deno} + 1 \Rightarrow \text{deno}$

//下一项分母的值

S8: 若  $\text{deno} \leq 100$  返回S4;

//给出累加结束条件

否则算法结束。

请思考：

如果第一项的值必须计算，则sum必须设初值

1)对于求和，sum的初值应设为多少？

2)对于求阶乘，sum的初值应设为多少？

## 2.3 算法的特性

算法的五个特性：

有穷性：一个算法必须在执行有穷步之后结束。

确定性：算法的每一步必须是确切定义的，对于相同输入必须得到相同结果。

有效性：算法的每一步都是能够实现的，即可操作的。

输入：算法有零个或多个输入。

有输出：算法执行完毕，必须有一个或若干个输出结果。

## 2.4 怎样描述算法

### 2.4.1 自然语言描述法

### 2.4.2 流程图表示

常用符号有：

ANSI

(American National  
Standard Institute)

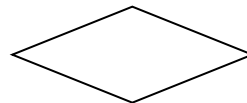
BS (a bowl of spaghetti)



起止框



输入/输出框



判断框

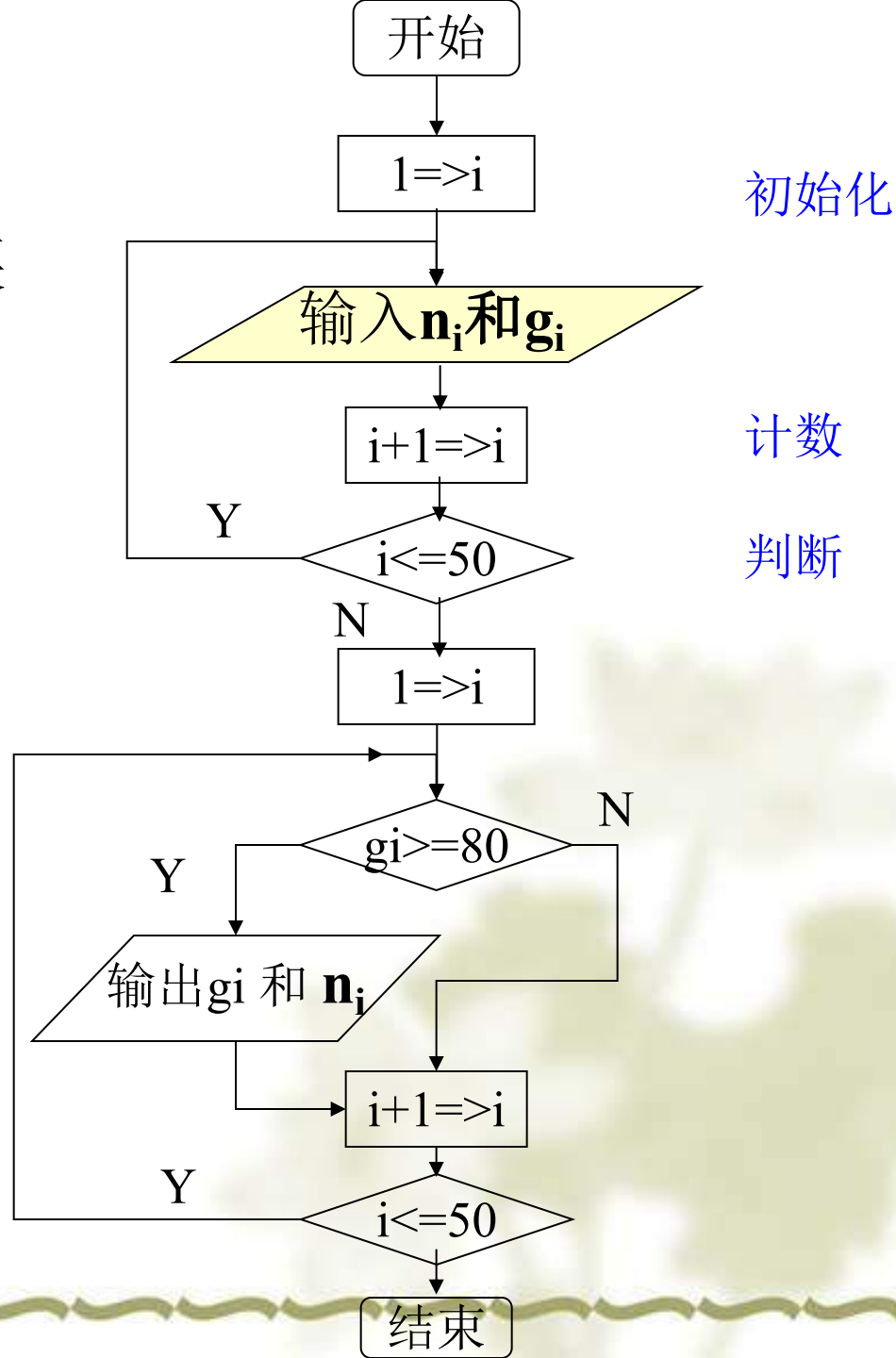


处理框



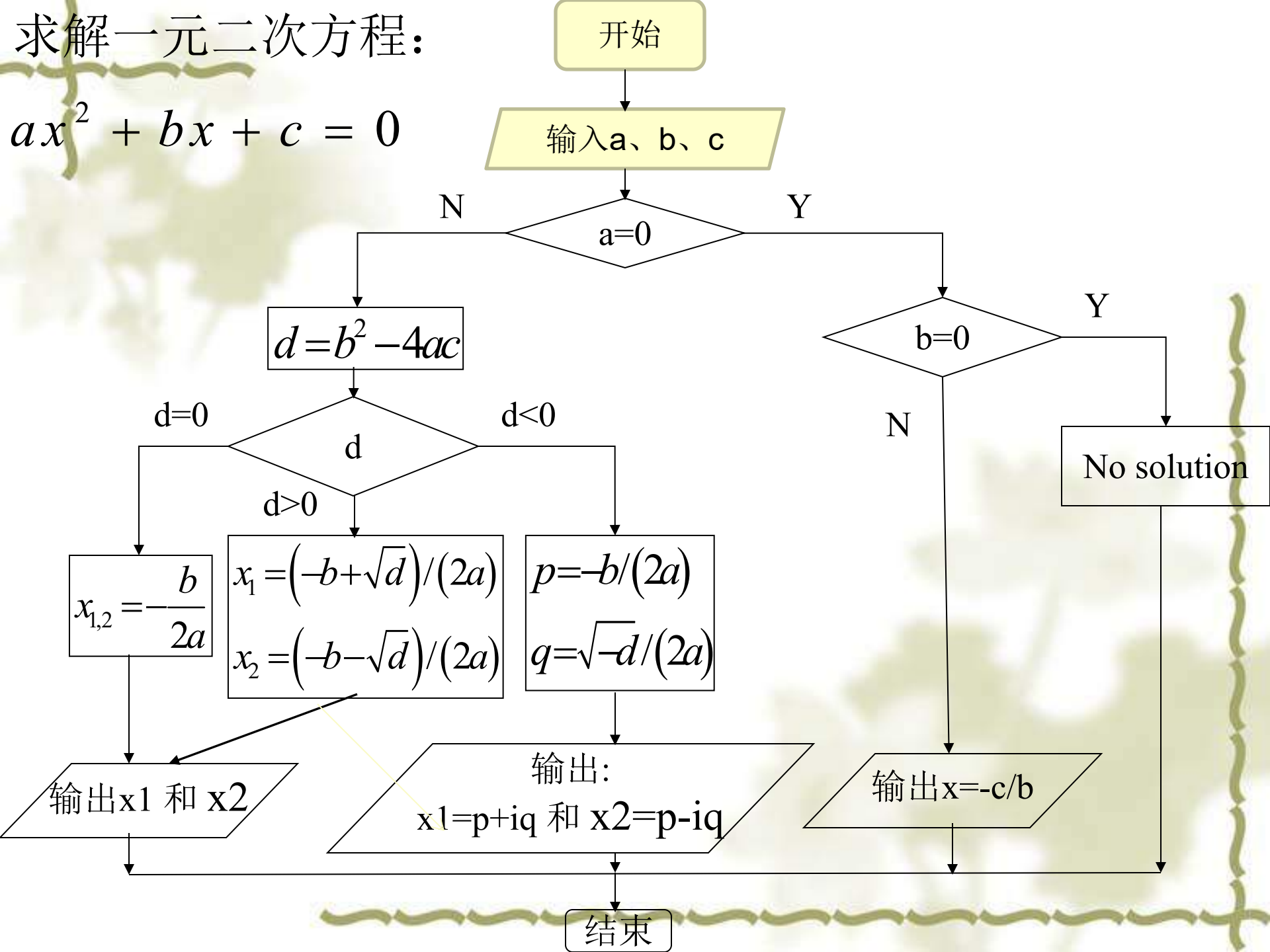
流程线

输入50个学生的学号和某门功课的成绩，并将其中成绩在80分以上学生的学号和成绩打印出来



求解一元二次方程：

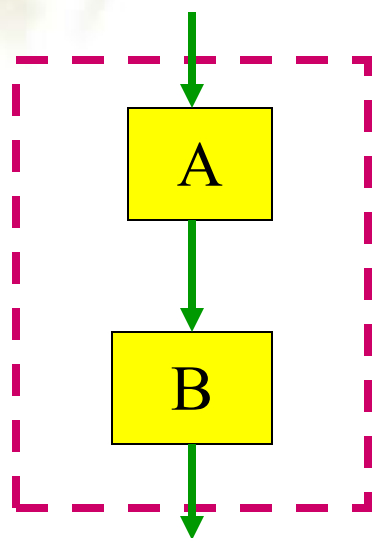
$$ax^2 + bx + c = 0$$



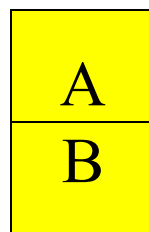


## 2.4.3 程序的三种基本结构和改进的流程图 (N-S结构流程图)

### 一、顺序结构



(a)



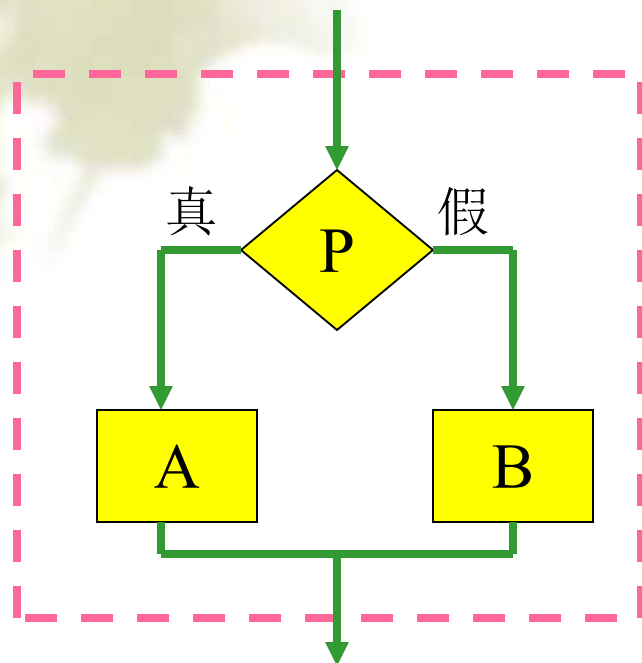
(b)

N-S结构流程图

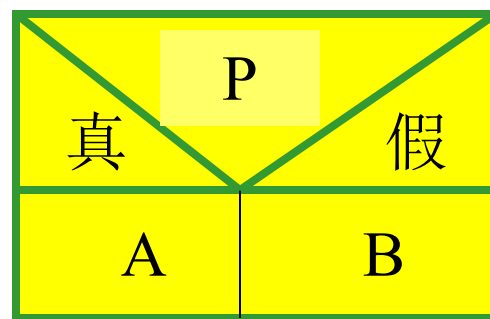
先执行A操作，再执行B操作，两者是顺序执行关系。



## 二、选择结构



(a)

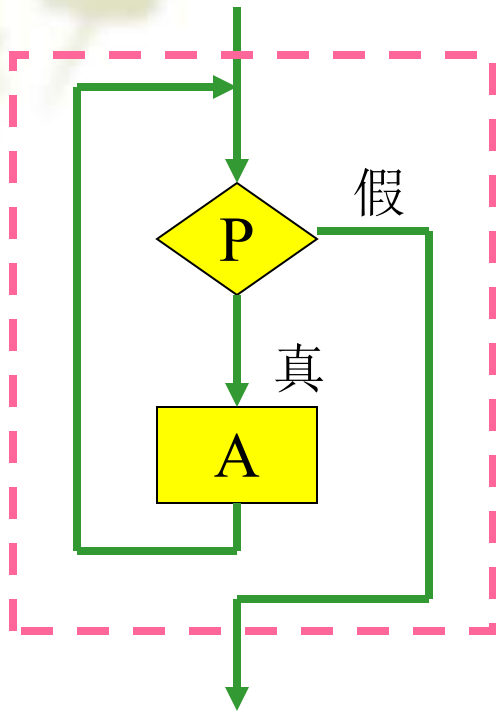


(b)

当P条件为真时，执行A模块，否则执行B模块。

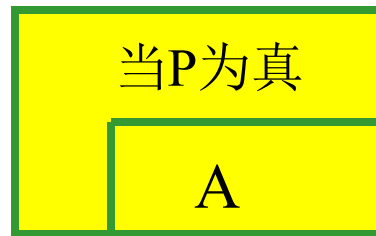
### 三、循环结构

#### 1. 当型循环结构



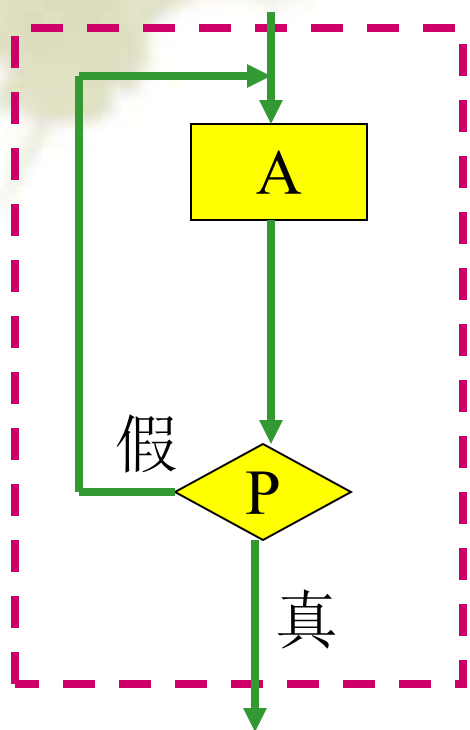
(a)

当P条件成立时，反复执行A,直到P为假。



(b)

## 2. 直到型循环结构

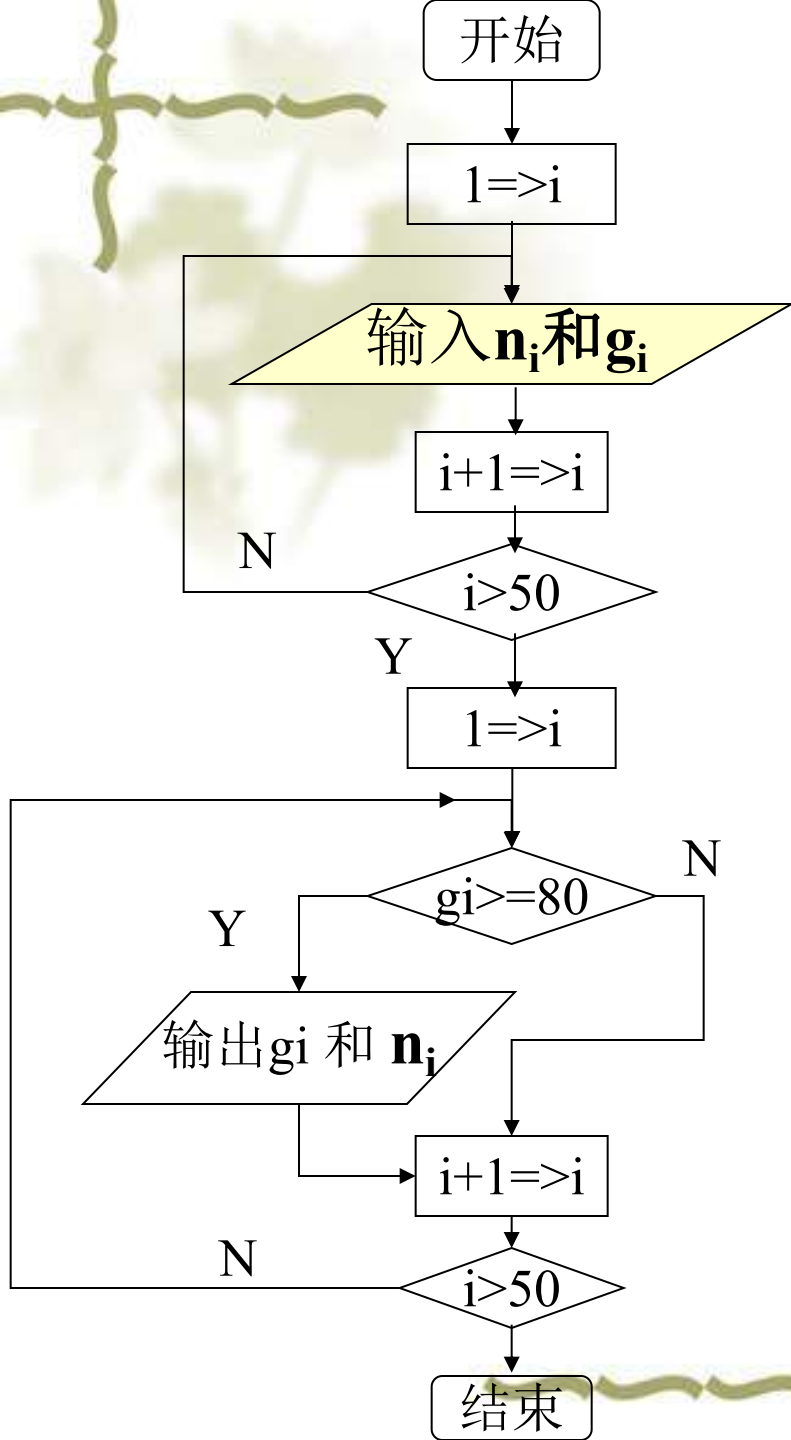


(a)



(b)

先执行A操作，再判断P是否为假，若P为假，再执行A，直到P为真为止。



## 2.4.5 用伪代码表示算法：用介于自然语言和计算机语言之间的文字和符号来描述算法（pseudo code）

**BEGIN**

**1**  $\Rightarrow i$

while( $i \leq 50$ )

{ input  $n_i$ 和 $g_i$

$i+1 \Rightarrow i$  }

**1**  $\Rightarrow i$

while( $i \leq 50$ )

{ if ( $g_i \geq 80$ ) print  $n_i$ 和 $g_i$

$i+1 \Rightarrow i$  }

**END**

## 2.4.6 用计算机语言表示算法

例4：求  $1 - 1/2 + 1/3 - 1/4 + \dots + 1/99 - 1/100$ 。

S1:  $1 \Rightarrow \text{sign}$

S2:  $2 \Rightarrow \text{deno}$

S3:  $1 \Rightarrow \text{sum}$

S4:  $(-1) * \text{sign} \Rightarrow \text{sign}$

S5:  $\text{sign} * (1/\text{deno}) \Rightarrow \text{term}$

S6:  $\text{sum} + \text{term} \Rightarrow \text{sum}$

S7:  $\text{deno} + 1 \Rightarrow \text{deno}$

S8: 若  $\text{deno} \leq 100$  返回S4；否则算法结束。

```
main()
{
    int sign=1;
    float deno=2.0 , sum=1.0, term;
    while (deno<=100)
    {
        sign= -sign;
        term=sign/deno;
        sum=sum+term;
        deno=deno+1;
    }
    printf("%f",sum);
}
```

## 2.5 结构化程序设计方法

### 1. 自顶向下

**Vs**自下而上，逐步积累

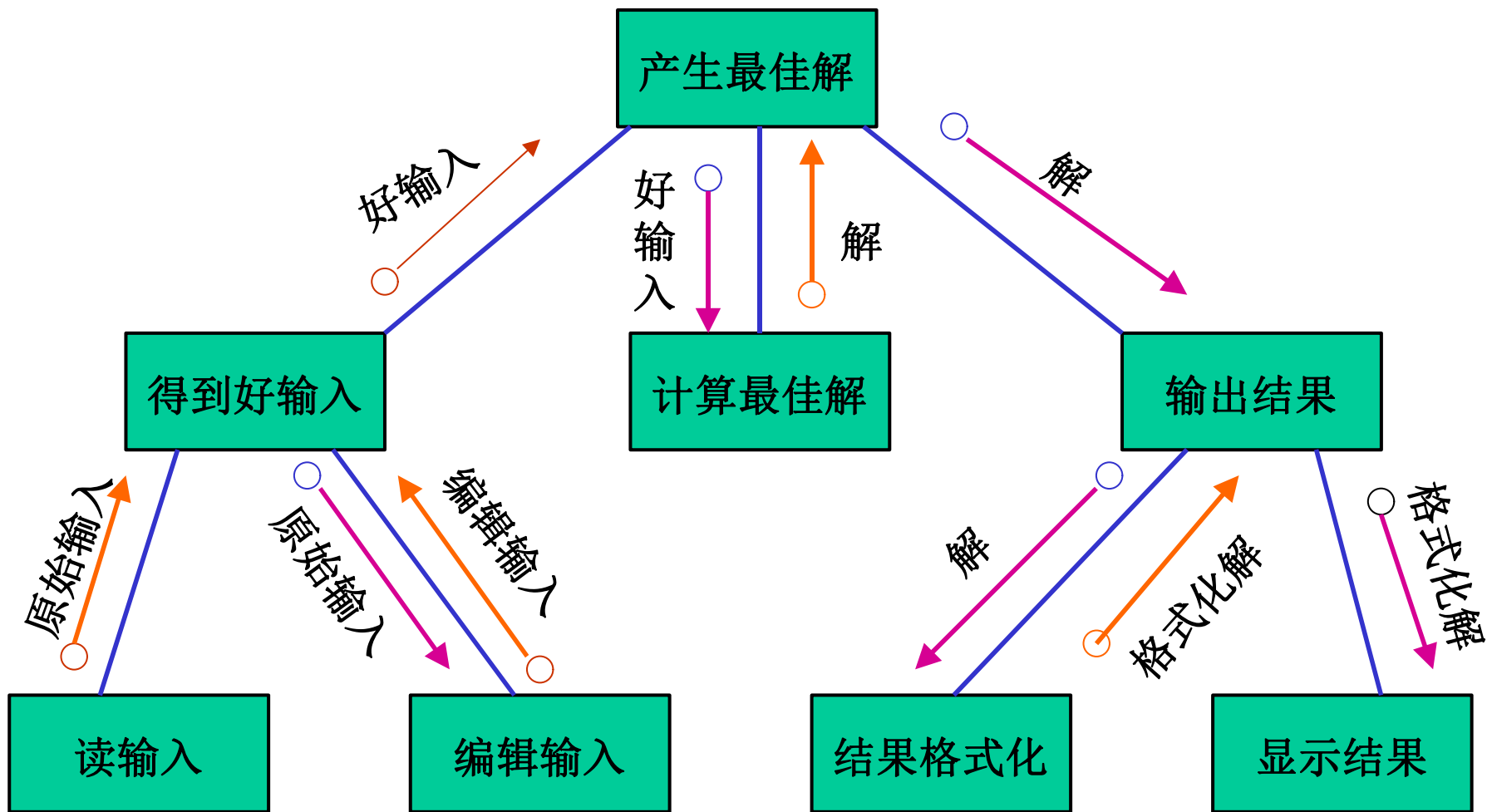
### 2. 逐步细化

### 3. 模块化设计

分而治之

### 4. 结构化编码

用高级语言正确地实现三种基本结构



产生最佳解的一般结构



```
main()
```

```
{
```

得到好输入



计算最佳解



输出结果

```
}
```

得到好输入

读输入

编辑输入

**RETURN(编辑好的输入)**

读输入

{

输入处理;

return 输入;

}

编辑输入

{

编辑输入处理;

return 编辑好的输入;

}

计算最佳解

```
{  
    求解处理;  
    return (解);  
}
```

结果格式化

```
{  
    结果格式化处理;  
    return 格式化结果;  
}
```

输出结果

结果格式化

显示结果

显示结果

```
{  
    显示结果处理;  
}
```

# 要牢记

- ❖ 每一个C程序需要一个main()函数（使用多于一个main()函数是非法的），“main”所在的位置是程序开始执行的地方。
- ❖ C程序**必须**以小写字母编写。然而，大写字母可以用于变量名和输出字符串的情况。
- ❖ 程序行中的所有单词必须用至少一个空格、制表符(tab)或者标点符号分隔开来。
- ❖ C程序的每一条语句必须用分号“;”结尾。
- ❖ 所有变量在使用前必须声明其类型和变量名。

# 要牢记

- ❖ 当程序使用一些特殊的名字或函数，而它们又没有在程序中定义时，要使用**#include**指令将有关的头文件包含到本程序中来。
- ❖ **include**和**define**是编译器**指令**而非C程序的**语句**，**不需要**用分号“；”结尾。
- ❖ 编译器指令的**#**符号必须放在该指令行的第一列。
- ❖ 花括号常用于将多条语句组合成一个整体，形成所谓**语句块**。花括号要成对“{ }”使用。
- ❖ 说明文字可以放在程序的任何空格处，用于增加程序的可读性和可理解性，“/\* \*/”在程序中也要成对出现。

# 第1次作业(1)

❖ 书上P36 2.4 (1), (2), (4)

(可以用Word的**插入**=>**形状**功能插入流程图的各种框图。实在不会用，用手画也行)

# 第1次作业(2)

—以下无需提交，自己对答案

看书，掌握用Turbo C(TC)运行C程序的步骤。

1\_1 判断下列描述是对还是错

- a) 每一个C程序以文字“END”结束。
- b) `main()`是程序开始执行的地方。
- c) 程序中一行可以有大于一条的语句。
- d) 一个`printf`语句只能产生一行输出。
- e) `main()`函数的右花括号是程序的逻辑结束。
- f) 诸如`stdio.h`的头文件的目的是用于储存程序的源代码。
- g) 注释语句让计算机把包括在`/* */`之间的文字打印输出。
- h) 语法错误(syntax error)由编译器检查出来。

# 第1次作业(3)

1\_2 下列哪一个描述是对的？

- a) 每一个C程序有至少一个用户定义函数。
- b) 只有一个函数可以命名为main()。

1\_3 下列哪一个关于注释的描述是错的？

- a) 注释的使用降低了程序运行的速度。
- b) 注释可以插入在一条语句的中间。
- c) 注释可以用作编程员的内部文件。

1\_4 在下列描述的空白处填入合适的文字。

- a) C程序的每一条语句必须以\_\_\_\_\_结束。
- b) \_\_\_\_\_函数用于将输出显示在屏幕上。
- c) \_\_\_\_\_头文件包含有标准数学函数。



# 第1次作业(4)

1\_5 如果有的话，找出下列程序中的错误

a) `/* A simple program`

```
int main()
```

```
{
```

```
/*Does nothing*/
```

```
}
```

b) `#include (stdio.h)`

```
void main(void)
```

```
{
```

```
    print("Hello C");
```

```
}
```

c) `include<math.h>`

```
main{
```

# 答案

1\_1

- a) False, “}”
- b) True
- c) True
- d) False, 可以用格式“\n”换行
- e) True
- f) False, 用于将常用函数包含到程序中来
- g) False, 用于标出注释语句
- h) True

1\_2

- a) False
- b) True

1\_3

- a) False
- b) True
- c) True

1\_4

- a) ;
- b) printf()
- c) math.h

1\_5

# 考试模拟题

## ❖ 选择题

1. 以下C语言标识符中不正确的是\_\_\_\_。  
A) int B) a\_1\_2 C) ab1exe D) \_x
2. 以下C语言标识符中正确的是\_\_\_\_。  
A) #define B) \_123 C) %d D) \n
3. C语言程序从main()函数开始执行，所以这个函数要写在\_\_\_\_。  
A) 程序文件的开始 B) 程序文件的最后  
C) 它所调用的函数的前面 D) 程序文件的任何位置
4. 下列关于C语言的叙述错误的是\_\_\_\_  
A) 大写字母和小写字母的意义相同  
B) 不同类型的变量可以在一个表达式中  
C) 在赋值表达式中等号(=)左边的变量和右边的值可以是不同类型  
D) 同一个运算符在不同的场合可以有不同的含义

# 考试模拟题

5. 一个C程序的执行时从\_\_\_\_

- A) 本程序的main函数开始，到main函数结束
- B) 本程序文件的第一个函数开始，到本程序文件的最后一个函数结束
- C) 本程序的main函数开始，到本程序文件的最后一个函数结束
- D) 本程序文件的第一个函数开始，到本程序main 函数结束

6. 以下叙述正确的是\_\_\_\_

- A) 在C 程序中，main 函数必须位于程序的最前面
- B) C 程序的每行中只能写一条语句
- C) C 语言本身没有输入输出语句
- D) 在对一个C 程序进行编译的过程中，可发现注释中的拼写错误

7. 以下叙述不正确的是\_\_\_\_

- A) 一个C 源程序可由一个或多个函数组成
- B) 一个C 源程序必须包含一个main 函数

# 考试模拟题

- C) 程序的基本组成单位是函数
- D) 在C 程序中，注释说明只能位于一条语句的后面
- 8. 一个C 语言程序是由\_\_\_\_
  - A) 一个主程序和若干子程序组成
  - B) 若干过程组成
  - C) 函数组成
  - D) 若干子程序组成

## ❖ 填空题

1. C 源程序的基本单位是\_\_\_\_\_.
2. 一个C源程序至少应包括一个\_\_\_\_\_.
3. 在一个C源程序中，注释部分两侧的分界符分别为\_\_\_\_\_和\_\_\_\_\_.
4. 在C语言中，输入操作是由库函数\_\_\_\_\_完成的，输出操作是由库函数\_\_\_\_\_完成的.

# 考试模拟题

## ❖ 编程题

1. 用传统流程图表示求解如下问题的算法：

已知银行整存整取存款不同期限的月息利率分别为：

0.315% 期限一年

0.330% 期限二年

0.345% 期限三年

0.375% 期限五年

0.420% 期限八年

要求输入存钱的本金和期限，求到期时能从银行得到的利息与本金的合计。

2. 用传统流程图表示求解如下问题的算法：

输入年份`year`和月`month`，求该月有多少天。判断是否为闰年。



# 考试模拟题答案

## ❖ 选择题

1.A 2.B 3.D 4.A 5.A 6.C 7.D 8.C

## ❖ 填空题

1.函数 2.主函数（或main函数） 3./\* 和 \*/

4.scanf printf

## ❖ 编程题



# 第三章 数据类型、运算符和表达式

## Chapter Three

### Data Types, Operator Symbols and Arithmetic Expressions

程序中使用的各种变量都应预先加以说明，即先说明，后使用。对变量的说明可以包括三个方面：

- 数据类型
- 存储类型
- 作用域

在本章中，我们只介绍数据类型说明。其它说明在以后各章中陆续介绍。

所谓数据类型是按被说明量的性质，表示形式，占据存储空间多少，构造特点来划分的。

在C语言中，数据类型可分为：**基本数据类型，构造数据类型，指针类型，空类型**四大类。

# 3.1 C语言数据类型



# C字符集

Letters字母	A.....Z	Uppercase	大写字母
	a.....z	Lowercase	小写字母
Digits数字	0.....9	All Decimal digits	数字
Special Characters 特殊符号	_	Under score	下划线
	,	comma	逗号
	.	period	小数点, 句号
	;	semicolon	分号
	:	colon	冒号
	?	Question mark	问号
	'	apostrophe	单引号
	''	quotation	双引号

## C字符集（续）

*	asterisk	星号
-	Minus sign	减号
+	Plus sign	加号
<	Opening angle bracket (or less than sign)	左尖括号 (或小于号)
>	closing angle bracket (or more than sign)	右尖括号 (或大于号)
(	Left parenthesis	左圆括号
)	Right parenthesis	右圆括号
[	Left bracket	左方括号
]	Right bracket	右方括号
{	Left brace	左大（花）括号
}	Right brace	右大（花）括号
#	Number sign	井号

## C字符集（续）

!	Exclamation mark	感叹号
	Vertical bar	竖线
/	slash	斜线
\	backslash	反斜线
~	tilde	波浪线
\$	Dollar sign	美元符号
%	Percent sign	百分号
&	ampersand	与
^	caret	幂次符号

## 3.2 常量与变量(Constant and Variable)

### 3.2.1 常量(Constant)

1. **定义:**在程序执行期间,其值不发生变化的量称为常量

2. **类型:**

❖ **直接常量(Direct ~)** 又称 **字面常量**。分为

- ✓ 整型常量: 123, -321, 0, 654321, +78
- ✓ 实型常量: 2.68, 3.14, 2.718, 1.38e12
- ✓ 单字符常量: 'a', 'C', 'z'
- ✓ 字符串常量: "china", "student", "class number"



## 3.2 常量与变量 (Constant and Variable)

- ❖ **符号常量 (Symbol ~)** 即用一个符号代表一个常量

例如: `#define PI 3.1415926` //预处理命令

```
main( )  
{  
    float r,l,s;  
    r=2;    l=2*PI*r; s=PI*r*r;  
    printf("l=%d, s=%d", l,s);  
}
```

编译器编译后，程序中的有关代码变成了（.obj文件，不可见）

```
l=2* 3.1415926*r;  
s= 3.1415926*r*r;
```



## 说明:

- ❖ 符号常量必须先定义后才能用它表示一个数值。
- ❖ 符号常量的值在其作用域内不能改变，也不能被再赋值。
- ❖ 符号常量名习惯用大写字母表示。

## 3.2.2 变量

1. **定义**：程序执行期间值可以改变的量。
2. **命名规则**：变量名由标识符表示，只能由字母、数字和下划线三种字符组成，且第一个字符不得为数字。

例如：下列标识符中，不合法的变量名有：

M.D.John , 12%gf , 1add , age&

5thclass, lotus-1-2-3 , cd\*ef , float

3. **变量要“先定义，后使用”**

定义格式：类型说明符 标识符1, ... , 标识符n;

例如：

int x, y, z; （每个整型变量分配 2字节bytes存储单元）

float a,b,c,d; （每个单精度变量分配 4字节存储单元）

double u,v,w; （每个双精度变量分配 8字节存储单元）

a	
	3

在C语言中，变量之所以要强制定义，其目的：

- 1、避免在使用时输错： `int student; stadent=30;`。
- 2、每一个变量被指定为一确定的类型，在编译时就能为其分配相应的存储单元。
- 3、指定每一个变量为一确定的类型，在编译时据此检查该变量所进行的运算是否合法。

例如：如果有

```
double x,y ; int z;
```

对于运算：

```
z=x*y;
```

在编译时系统就会提示该运算类型不匹配：

warning C4244: '=' : conversion from 'double' to 'int', possible loss of data

# 请 注 意

- 选变量名时，要做到“见名知意”

例如：name    student\_num    max    grade\_1    sum

- 大小写字母是两种不同的字符，C变量名习惯用小写字母表示。
- 变量名的长度因系统而异。当用TC编译系统时，不要超过8个字符，而用Visual C++编译系统时，几乎无限制。

## 3.3 整型数据

### 3.3.1 整型常量的表示法

**十进制整数：** 由数字（0-9）和正负号表示

例如：123、-456、0 等

**八进制整数：** 以0开头，含数字0~7

例如：037,0,0435,0551,0123 即  $(123)_8 = (83)_{10}$

$$(123)_8 = 1 \times 8^2 + 2 \times 8^1 + 3 \times 8^0 = (83)_{10}$$

**十六进制整数：** 以0x或0X开头，含数字和字符

16: 0~9, a, b, c, d, e, f, 10

10: 0~9, 10, 11, 12, 13, 14, 15, 16

例如：0X2,0x9F,0Xbcd,0X

0x123 即  $(123)_{16} = (291)_{10}$

-0x12 即  $(-12)_{16} = (-18)_{10}$

### 3.3.2 整型变量

1. 整型数据在内存中的存放形式（补码表示法：便于加  
减运算）

int i;

i=10;

$10=(1010)_2$

$$2 \mid 10 \quad 0 \times 2^0$$

$$2 \mid 5 \quad 1 \times 2^1$$

$$2 \mid 2 \quad 0 \times 2^2$$

$$\text{首位} \quad 1 \quad 1 \times 2^3$$

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

10的表示法,正数的补码等于它的原码

-10的  
表示

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

10的原码

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

取反

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

再加1,得-10的补码

计算10-10:

$$10-10=10+(-10)$$

10

0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

+

-10

1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

=

0

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---



## 2 整型变量的分类

1. 类型 {
- 基本型(16位): 以 `int` 表示  
(-32768~32767 :  $-2^{15}$ , 0,  $+2^{15}-1$ )
  - 短整型(16位): 以 `short int` 或 `short` 表示
  - 长整型(32位): 以 `long int` 或 `long` 表示  
( $-2.1 \times 10^9 \sim 2.1 \times 10^9$ )
  - 无符号型 {
    - 无符号基本型(`unsigned int`)
    - 无符号短整型(`unsigned short`)
    - 无符号长整型(`unsigned long`)

一个无符号整型变量中可以存放的正数的范围比一般整型变量中正数的范围扩大一倍。

```
unsigned int b;    /* 变量b的数值范围: 0~65535 */
```



## 整型变量的字长

short型： 2个字节byte（字长16位bit）

int 型： 一般是2个字节

long 型： 4个字节（字长32位）

C要求 short型数据长度短于int型， int型短于long型。  
( $\text{short} \leq \text{int} \leq \text{long}$ )

long 型可以得到大范围的数据（-20亿~20亿），但同时会降低运算速度，因此除非不得已，不要大量使用long型变量。

整型数据的溢出：

```
main()  
{ int a, b;  
a=32767; b=a+1;  
printf(“%d, %d”, a, b);
```

运行结果： 32767, -32768

改进：将变量a和b改成long型，并按 %ld 格式输出。

但有时，系统（例如Visual C++）自动将int型变量（16位）扩展为long型变量（32位）

## 整型数据的数据范围

请看“ANSI标准定义的整数类型”(P43表)。

注意：

1、在一个整数常量后面加一个字母l或L，

则认为是long 型常量。例如

123456789l,    9876543L

2、在一个整数常量后面加一个字母u或U，

则认为是unsigned int 型常量。例如

56789U,        56789u

## 3.4 实型数据 (Real Data)

### 3.4.1 实型常量

表示形式有如下两种：

**十进制数形式：**由数字和小数点及正负号组成。

(注意：必须有小数点)

例如：23.678, 0.0083, -0.75, 435.36, +247.00

也可能忽略小数点前面或者后面的数字

215., .95, -.71, +.5, .678 23.

**指数形式：**由数字、小数点、字母e或E及正负号组成。

(注：e或E之前必须有数字，其后指数必须为整数)

例如：3.5E-5 ( $3.5 \times 10^{-5}$ )      2e3 ( $2 \times 10^3$ )

错误：E2, 3.6e3.5, .e6, e

## 3.4.2 实型变量

实型变量可分为：

单精度（float型） 如： float x,y;

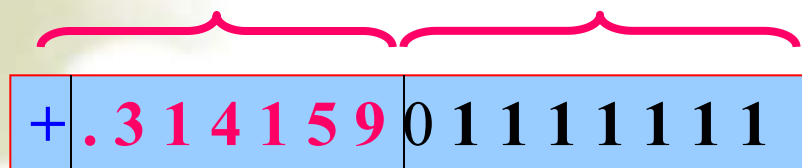
双精度型（double型） 如： double a, b, c;

说明：

1. float 型变量用于存储单精度实数，在内存中占4个字节(bytes)或32位(bits)，提供6~7位有效数字。
2. double 型变量用于存储双精度实数，在内存中占8个字节(bytes) 或64位(bits)，提供15~16位有效数字。

实数的存储方式与整形数据不同：

符号 小数部分 指数部分



$$3.14159 \times 10^{255}$$

对于float型10进制数，一些编译系统以24位表示小数部分，8位表示指数部分。

3. 一个实型常量不分float 和 double 型，一个实型常量可以赋给一个float 型或 double 型变量，并根据变量的类型截现实型常量中相应的有效位数字。

例如：

```
main()
```

```
{ float a,c ; double b;
```

```
  a=123456789012345.6789 ;
```

```
  b=123456789012345.6789 ;
```

```
  c=a+20;
```

```
  printf(“a=%f\nb=%f\nc=%f\n ”, a,b,c);
```

```
}
```

输出结果：

a=123456788103168.000000

b=123456789012345.670000

c= 123456788103188.000000

(避免将一个很大的数与一个很小的数相加减，以免出现舍入误差)



## 3.5 字符型数据 (Character Data)

### 3.5.1 字符常量 (Character Constant)

1. 用单引号括起来的一个字符

如 ‘**T**’ , ‘**7**’ , ‘**!**’

2. 转义字符: 是一种特殊的字符常量, 由反斜杠 ‘\’ 和一个字符组成, 其作用是用 ‘\’ 将后面的字符转变为另外的意义。

如 ‘\n’ 中的 ‘n’ 不表示字母n, 而表示“换行”的意思。这是一种控制字符。



例如:

字符 含义

‘\n’

换行

‘\t’

横向跳格（水平制表符）

‘\b’

退格

‘\v’

竖向跳格（垂直制表符）

‘\r’

回车

‘\f’

走纸换页

‘\a’

报警声

‘\0’

空字符

‘\’

反斜线

‘\’

单引号

‘\’

双引号

说明：

1、“\ddd”与“\xhh”分别表示用八进制数或十六进制数表示一个字符。

例: '\101'代表ASCII码(十进制)为65的字符 'A'. 系统中所有字符都有ASCII值 (见374页)

2、由于 \ ' " 在C语言中都有了特殊的意义，故要想在字符串中使用它们，须在其前面再加上一个反斜线。

[illegible]

例如, Tom said to the waiter: "I'd like a cup of coffee".

```
printf ("Tom said to the waiter: \"I'd like a cup of coffee\");
```

### 3、除了前面介绍的特殊转义字符外，‘\字母’

代表字母本身。如 ‘\c’ 代表字母 c。

## 3.5.2 字符变量(Character Variable)

定义形式: `char` 标识符1, 标识符2, ... , 标识符n

例如: `char c1, c2, c3, ch;`  
`c1='a'; c2='b'; c3='c'; ch=' d';`

说明:

- 1、字符变量只能存储一个字符，在内存中占一个字节。
- 2、在内存中，是把字符对应的ASCII码值放到存储单元中(0~255)。
- 3、字符型数据与整型数据之间可以通用。

```
main()
{ char c1,c2;
  c1=97;//数字赋值给字符变量
  c2=98;
  printf("%c %c\n",c1,c2);
  printf("%d %d\n",c1,c2);
}
```

输出结果:

```
a  b
97 98
```

```
main()
{ int i; char c;
  i='a'; //字符赋值给整型变量
  c=97; //数字赋值给字符变量
  printf("%c,%d\n",c,c);
  printf("%c,%d\n",i,i);
}
```

输出结果:

```
a, 97
a, 97
```

### 3.5.4 字符串常量(Character string constant)

1. 定义：括在一对双引号 “...” 之内的字符序列或转义字符序列称为字符串常量。

例如：“How are you!” “a” “\*abc\n\t”

2. 存储：字符串中的字符依次存储在内存中一块连续的区域内，并且把空操作字符 ‘\0’ 自动附加到字符串的尾部作为字符串的结束标志。故字符个数为  $n$  的字符串在内存中应占  $n+1$  个字节。

例：char c[6] = “china”;  
char \*s = “china”;

c	h	i	n	a	\0
---	---	---	---	---	----

‘b’ 与 ‘B’是否相同？

b为98， B为66

‘b’ 与 “b”是否相同？

‘b’是字符常量， 占据个1字节；

98	
----	--

“b”是字符串常量， 占据个2字节

98	0
----	---

‘b’ ‘\0’

## 3.6 变量赋初值 (Variable initialization)

1. *初始化*: 在定义变量的同时为变量赋初值
2. *形式*: 类型标识符    变量名=常量或常量表达式

例如:     **int    x=10 ;double sum=0.;**

**char   ch='a' ;**

- ❖ 变量赋初值允许使用符号常量

例如:   **#define PI 3.1415926**

**.....float   x=PI ; .....**

- ❖ 可对被定义的变量的一部分赋初值

例如:    **int   a , b, c=1, d=2;**

**float   r=2 ,l, s;**

- ❖ 可对几个变量赋以同一个初值

例如:    **int   a=6, b=6, c=6;**

或可写为:   **int   a=b=c=6;**



# 数据类型的尺寸和范围

类型 type	字节 bytes	字长 bits	有效数字	取值范围
char	■	8		0~255
int	■ ■	16		-32768~32767
unsigned int	■ ■	16		0~65535
short int	■	8		-128~127
long int	■ ■ ■ ■	32		$-2.147 \times 10^9 \sim 2.147 \times 10^9$
float	■ ■ ■ ■	32	6~7	$-3.4 \times 10^{-38} \sim 3.4 \times 10^{+38}$
double	■ ■ ■ ■ ■ ■ ■ ■	64	15~16	$-1.7 \times 10^{-308} \sim 1.7 \times 10^{+308}$
long double	■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■	128	18~19	$-1.2 \times 10^{-4932} \sim 1.2 \times 10^{+4932}$



## 3.7 各类数值型数据间的混合运算

形式： 隐式转换 （由系统自动进行）

显式转换 （强制类型转换）

隐式转换的转换规则：

先自动转换成同类型数据，再进行运算。规则是低字长类型转换成高字长类型。

```
double (8 bytes) ← float (4 bytes)
    ↑
long int (4 bytes)
    ↑
unsigned int (2 bytes)
    ↑
int (2 bytes) ← char, short (1 bytes)
```

## 3.7 各类数值型数据间的混合运算

```
double (8 bytes) ← float (4 bytes)
    ↑
long int (4 bytes)
    ↑
unsigned int (2 bytes)
    ↑
int (2 bytes) ← char, short (1 bytes)
```

- (1) 箭头表示当运算对象类型不同时的转换方向，一般是从低精度的类型转换到高精度的类型。
- (2) **向左的横向箭头**表示即使在同一种数据类型间进行运算时也要进行转换，用于**提高精度**。即：
  - 同为float 型数据的运算，一律转换成double 型数据后进行；
  - 同为char 和short int 型数据的运算，一律转换成 int 型的数据后进行。

### 3.7 各类数值型数据间的混合运算

double (8 bytes) ← float (4 bytes)  
↑  
long int (4 bytes)  
↑  
unsigned int (2 bytes)  
↑  
int (2 bytes) ← char, short (1 bytes)

(3)当运算对象具有相同类型时，**向上的箭头**不再起作用。即：

- **int型与int 型数据运算的结果仍为int 型**；
- unsigned型与unsigned型数据运算的结果仍为unsigned型；
- long型与long 型数据运算的结果仍为long 型。

❖ char-char => int-int => char

'd' - 'a' => 100 - 97 => 003 => '♥'

❖ float+float => double+double => double  
int/float=>float/float=>float

float a; a=1/3.0=1.0/3.0=0.33333333

double c; c=1.0/3.0=0.333333333333333333

int b; b=1/3.0=1.0/3.0=0.33333333=0

❖ int/int=>int

float a; a=1/3=0.0;

int b; b=1/3=0;

Decision  
loss

例：若有： **int i ; float f ; double d ;**  
**long e ;**

试判断表达式 **3 + 'b' + i \* f - d / e** 的类型

$$\begin{array}{ccccccc} & & \underbrace{\hspace{1cm}} & & \underbrace{\hspace{1cm}} & & \\ & & \text{int} & & \text{float} & & \\ & & \underbrace{\hspace{1cm}} & & \underbrace{\hspace{1cm}} & & \\ & & \text{float} & & \text{double} & & \\ & & \underbrace{\hspace{2cm}} & & & & \\ & & \text{double} & & & & \end{array}$$

例：如果 **int a=5,b=2;**  
**double x;**  
**x=a/b;**  
**printf(" x=%f",x);**  
输出结果是什么？

**X=2.000000**

## C语言使用的基本表达式有：

(1)赋值表达式，如：  $a=3$

(2)算术表达式，如：  $3+4*2$

(3)关系表达式，如：

$3>2$ ,  $a==b$ ,  $c!=0$

(4)逻辑表达式，如：  $3>5 \&\& a>=b$

(5)条件表达式，如：  $a>b?a:b$

(6)逗号表达式，如：  $a=3, b=4, c=5$

# 3.8 算术运算符和算术表达式

## (Arithmetic Operators and Expressions)

### 1、基本的算术运算符:

加(+), 减(-), 乘(\*), 除(/), 求余(或称模运算, %)

整型数运算

`int a=14,b=4;`

`a-b=10`

`a+b=18`

`a*b=56`

`a/b=3`(decimal part truncated)

`a%b=2`(remainder of division)

`-14%3=-2`



- 实型数运算

float x,y,z;

$x=6.0/7.0=0.857143$

$y=1.0/3.0=0.333333$

$z=-2.0/3.0=-0.666667$

% operator can not be used here

- 混合运算

$15/10.0=1.5$

$15/10=1$  ?



说明：

- (1) 上述运算符均为双目运算符（即有两个操作数two operands）
- (2) “/” 运算中，若操作数均为整数，则执行整除运算，舍去小数部分。例如：5/3=1
- (3) “%” 要求两侧均为整数，“%” 运算不能用于float 和 double型数据

例如：     5 % 3 =    2

## 2、算术表达式和运算符的优先级与结合性 (Priority and Combination, 附录III )

(1) 算术表达式：用算术运算符和括号将操作数连接起来的，符合 C 语法规则的式子。

例如： ‘ a’ + ‘b’ + a\*b/c +2.8

(2) 优先级： C 语言中，运算符的运算**优先级**共分为 15 级，1 级最高，15 级最低。

✎ 在表达式中，优先级较高的先于优先级较低的进行运算。

\*   /   %   高于   +   -

✎ 而在一个运算量两侧的运算符优先级相同时， 则按运算符的**结合性**所规定的结合方向处理。

(3) 结合性：C语言中各运算符的结合性分为两种，即左结合性(自左至右)和右结合性(自右至左)。这种自左至右的结合方向就称为“左结合性”。

例如算术运算符的结合性是自左至右，即先左后右。如有表达式 $x-y+z$ 则 $y$ 应先与“-”号结合，执行 $x-y$ 运算，然后再执行 $+z$ 的运算。

例如： $a/b*c$

$$1/3*3=0$$

$$3*1/3=1$$

而自右至左的结合方向称为“右结合性”。

最典型的右结合性运算符是赋值运算符。如 $x=y=z$ ，由于“=”的右结合性，应先执行 $y=z$ 再执行 $x=(y=z)$ 运算。

### 3、强制类型转换(Forced type transformation)

形式：(类型名) (表达式)

例如：(double) (a+b)

说明：

(1) 表达式应用括号括起

例如：(int) (x+y);

(2) 在强制类型转换时，得到一个所需类型的中间变量，原来变量类型未发生变化。

(3) 用途：

✎ 完成某种运算，如(int)5.5%3;

✎ 在函数调用时使实参与形参一致。

举例

作用

`x=(int) 7.5`

7.5 通过截尾转换为整数

`a=(int)21.3/(int)4.5` 转换为21/4，结果为5

`b=(double)sum/n`

以浮点数（实型数）模式计算除法

`y=(int)(a+b)`

将表达式`a+b`转换为整数

`z=(int)a+b`

将`a`转换为整数然后与`b`相加

`p=cos((double)x)`

在使用 `x`之前，将其转换为实型数

注意： 强制类型转换后并不改变原来变量的类型

请看下面程序段：

```
main( )  
{  
    float x;  
    int i;  
    x=3.6;  
    i=(int)x;  
    printf(“x=%f, i=%d”,x,i);  
}
```

输出结果： **x=3.600000, i=3**

## 4、自增、自减运算符（++、--）

**作用：**使变量的值增 1 或减 1，如：

$\begin{cases} i++, ++i & \text{等于 } i=i+1 \\ i--, --i & \text{等于 } i=i-1 \end{cases}$

$\begin{cases} ++i, --i & \longleftrightarrow \text{先执行 } i=i+1(i=i-1), \text{后使用 } i \text{ 的值} \\ i++, i-- & \longleftrightarrow \text{先使用 } i \text{ 的值, 后执行 } i=i+1(i=i-1) \end{cases}$

• **举例：**

```
int j,i=3;
```

- ① `j=++i;` /\*j的值为4, i为4\*/
- ② `j=i++;` /\*j的值为3,然后i的值变为4 \*/
- ③ `j=-i++;` /\*j的值-3, 然后i的值变为4 \*/
- ④ `printf("%d",i++)` /\*输出i的值3, 然后i的值变为4 \*/



## 4、自增、自减运算符（++、--）

说明：（1）为单目运算符

（2）++、--只能用于变量，而不能用于常量或表达式。

例如：1 0 ++, (x+y)++, ++'a', b++ 中合法的为：

（3）自增、自减运算符为右结合性（即自右至左）。

例如：-a++  $\longleftrightarrow$  -(a++)

错误的结合 (-a)++

y\*x++  $\longleftrightarrow$  y\*(x++)

错误的结合 (y\*x)++

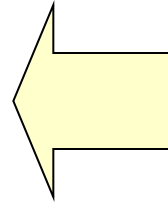


## 练习题

1. 设 $a=2$ , 则执行语句  $k=++a+a++$  后的  $k$  值为多少?  
( $k=6, a=4$ )

$a=4$   
 $a=3, 5 = 313$

2.  $\text{int } i=3;$   
 $a=(i++)+(i++)+(i++)$  ?
3.  $\text{int } i=3;$   
 $\text{printf}("%d, %d", i, i++)$  ?



请避免  
这种写法

P58

最好写成

```
i=3; j=i++;  
printf("%d, %d", i, j) ?
```

## 3.9 赋值运算符和赋值表达式

### Assignment Operator and Expressions

#### 1、 赋值运算符 (=)

形式：变量名=常量或表达式

作用：将右边常量或表达式的值赋给左边的变量

例如： `b=c+d;`

要求：如果表达式类型与左边的变量的类型不匹配，自动进行类型转换。

## 2、类型转换的原则

(1) 将实型数据赋给整型变量时,舍弃实数的小数部分。

如: 若a为int型变量, 执行a=3.56后, 则a=3

(2) 将整型数据赋给实型数据时,数值不变,但以浮点形式存储到变量中。

如: 若b为float型变量, 执行b=35后, 则b为35.00000

(3) 整型数据赋给字符型变量时,只将低8位原封不动地送到字符变量中(即截断)。

如:

0000 0001	0010 0001
-----------	-----------

i=289=256+33



0010 0001
-----------

c=33    c='!'

(4) 字符型数据赋给整型变量时, 将字符数据(8位)放到整型变量低8位中。高8位则需要依据“符号扩展”来决定。

例1 (无符号字符数据赋给整型变量)

```
unsigned char c;
```

```
int i;
```

```
c=254;
```

```
i=c;
```

1111 1110

c= 254



0000 0000

1111 1110

i=254

例2 (Turbo C 系统把字符处理为带符号的)

```
char c;
```

```
int i;
```

```
c=254;
```

```
i=c; /* 变量c 以整数形式输出为-2 */
```

1111 1110

c= 254

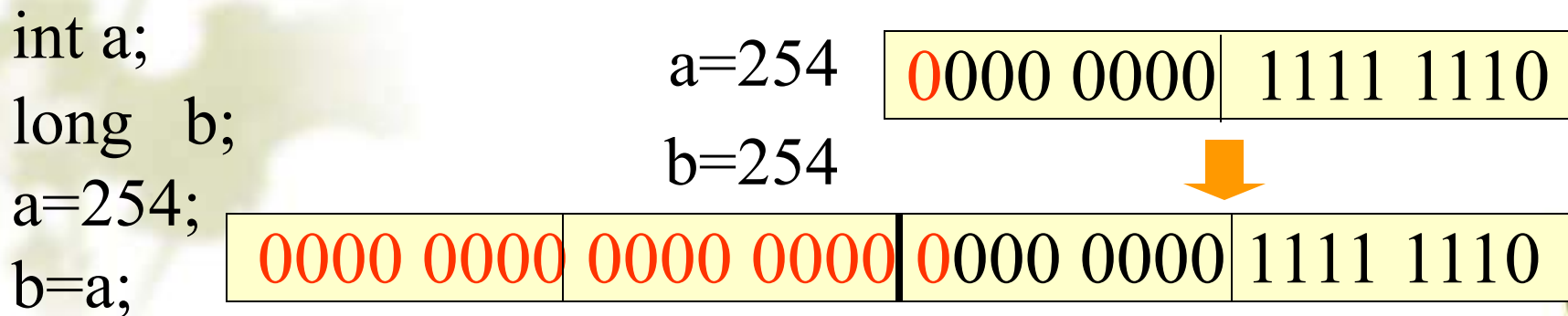


1111 1111 1111 1110

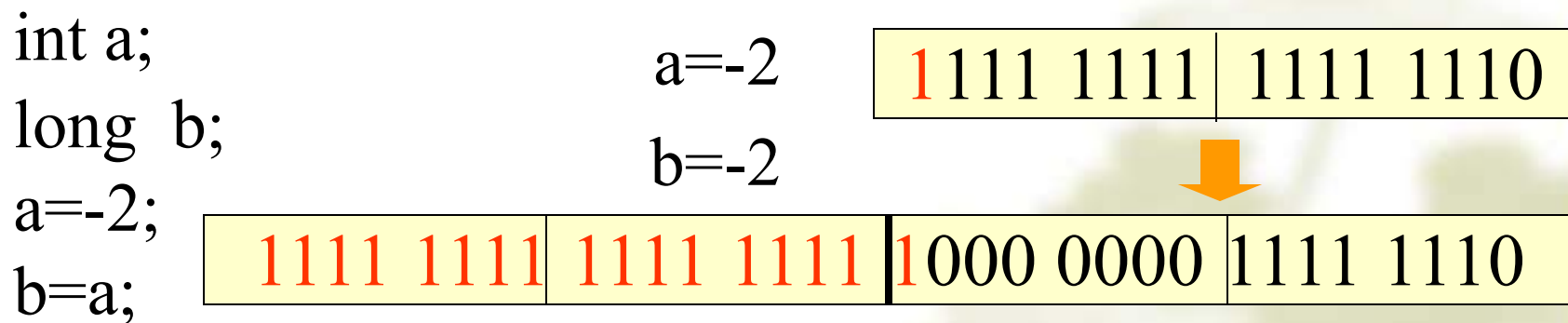
i=-2

- (5) 将带符号数据赋给长度相同的无符号型变量，原样照赋。
- (6) **double** 赋给 **float** 时，截取前面7位有效数字。

### 例3：（带符号int型数据赋给long型变量）



### 例4：（带符号int型数据赋给long型变量）



例5: (无符号int型数据赋给long型变量)

```
unsigned int a;
```

```
long b;
```

```
a=65534;
```

```
b=a;
```

a=65534

b=65534

1111 1111	1111 1110
-----------	-----------



0000 0000	0000 0000	1111 1111	1111 1110
-----------	-----------	-----------	-----------

例6: (有符号数据赋给无符号变量)

```
int a;
```

```
unsigned int b;
```

```
a=-2;
```

```
b=a;
```

a=-2

b=65534

1111 1111	1111 1110
-----------	-----------



1111 1111	1111 1110
-----------	-----------

P66 (3.11) 写出下面赋值的结果。格中写了数值的是要将它赋给其它类型的变量，将所有空格添上赋值后的数值。

int	99	100	76	53	68	42	-1
char	'c'	'd'	'L'	'5'	'D'	'*'	X
unsigned int	99	100	76	53	68	42	65535
float	99.000000	100.000000	76.000000	53.65	68.000000	42.000000	65535.000000
long int	99	100	76	53	68	42	65535



## ❖ 总的结论：

- ❧ 低精度的数据赋值给高精度的变量，其精度将保持；
- ❧ 高精度的数据赋值给低精度的变量，其精度将降低，有时甚至可能出错。

### 3、复合赋值运算符

#### (Compound assignment operators)

- 复合赋值运算符：在赋值符“=”之前加上其它运算符
- 共有10种复合赋值运算符

$+=$  ,  $-=$  ,  $*=$  ,  $/=$  ,  $\%=$  ,

$<<=$  ,  $>>=$  ,  $\&=$  ,  $\^{}=$  ,  $|=$  （位运算）

例如：  $a+=3$   $\longleftrightarrow$   $a=a+3$

$x*=y+3$   $\longleftrightarrow$   $x=x*(y+3)$

$x/=y$   $\longleftrightarrow$   $x=x/y$

## 普通赋值运算符

`a=a+3`

`a=a-3`

`a=a*(n+1)`

`a=a/(n+1)`

`a=a%b`

`a=b=c=6`

`a=(b=4)+(c=6)`

`a=(b=4)*(c=6)`

## 简洁赋值运算符

`a+=3`

`a-=3`

`a*=n+1`

`a/=n+1`

`a%=b`

复合赋值符这种写法，对初学者可能不习惯，但十分有利于编译处理，能提高编译效率并产生质量较高的目标代码。

## 4、赋值表达式(Assignment expressions)

定义：由赋值运算符将一个变量和一个表达式连接起来的式子。

形式：变量 = 表达式

结合方向：由右向左

说明：赋值表达式的值为被赋值的变量的值。其中“表达式”又可以是一个赋值表达式。

例如：a=b=c=6

a=(b=4)+(c=6)

a=(b=4)\*(c=6)

x=-b/(2\*a)

y1=-x+(dt=sqrt(b\*b-4\*a\*c))

y2=-x-dt

- 赋值表达式中可以包含复合赋值运算符

例如：若 $a=12$ ,计算表达式

$$a+=a-=a*a$$

运算后 $a$ 的值

$$\text{第一步: } a-=a*a \iff a=a-a*a=12-12*12=-132$$

$$\text{第二步: } a+=-132 \iff a=a+(-132)=-264$$

错误理解:  $a=a+(a-=a*a)=a+a-a*a=-120$

❖ 练习:

写出顺序执行下列表达式运算后的 $x$ 值:

$$1. \ x=a=b=10 \quad X=10$$

$$2. \ x=25\%(c=3) \quad X=1$$

## 3.10 逗号运算符和逗号表达式

1.逗号运算符：即“，”

优先级：为所有运算符中优先级别最低的。

### 2.逗号表达式

形式：表达式1，表达式2，……，表达式n

求解过程：顺次求解表达式1，表达式2，……，最后求解表达式n，逗号表达式的值为表达式n的值。

例如：a=3\*5, a\*4 60

a=3\*5, a\*4, a+5 20

x=(a=4%3, a+1, a\*10) 10

a=2,b=3,c=4;

例如：

```
v=(x=10, y=5, x+y);
```

The value of v is 15.

应用：

1) 在for循环中：

```
for(n=1, m=10; n<=m; n++, m++)
```

2) 在while循环中：

```
while(c=getchar(), c!='\0')
```

3) 数值交换：

```
t=x, x=y, y=t;
```

程序中使用逗号表达式，通常是要分别求逗号表达式内各表达式的值，并不一定要求整个逗号表达式的值。



# 要牢记

- ❖ 不要使用关键词或者系统库函数的名字作为标识符（变量名或函数名）；
- ❖ 不要使用只差一、两个字符的变量名；
- ❖ 每个变量要在程序或函数开始处声明其类型；
- ❖ 在程序中被使用前，所有变量必须赋值；
- ❖ 不要在**#define**命令的结尾处加“;”；
- ❖ 不要在**#**和**define**之间加空格；
- ❖ **C**对于数据的溢出不给任何警告或指示，只会给出错误的结果。必须小心定义数据的类型；



## 要牢记

- ❖ 小心使用自增和自减运算符，在使用前了解前置（如++i）和后置（如i++）的差别；
- ❖ 在计算顺序混乱的地方，小括号的使用可使它们变清晰；
- ❖ 在赋值以前，表达式的类型被转换成与等号左边的变量类型一致。注意转换时可能的数据损失；
- ❖ 所有的数学函数的参数（自变量）和返回值（计算值）均为double类型（例如y=sin(x)中的x和y）；
- ❖ 不要对浮点数变量使用自增和自减运算符。

### 第三章 作业（所有程序均需上机调试，正确后方能提交）

3\_1 判断以下程序段有否错误，写出正确的程序段，并上机调试验证你的程序。（提交修改后的源程序）

```
1      #include<stdio.h>;
2      void mian{}
3      {      float x=3.1415;
4              int y,z,student_num=196;
5              PRINTF("Hello!\n");
6              printf("the student_number is:d%\n",num);
7              printf("the student_number is:%D\n", student_num);
8              printf("x=%d\n",x);
9              printf("y=%f\n",y);
10             z=3/2;
11             printf("z=%d\n",z);
12     }
```

//说明:”%d”是输入、输出整形数据的格式, ”%f”是输入、输出实形数据的格式, ”\n”是换行的格式

3\_2 在计算机屏幕上输出一段文字：

John said to his brother:(换行)

(空两格)"Let's have a rest!"

并在此段文字前后各空两行。（提交源程序）

3\_3 求下面算术表达式的值：

(1)  $x+a\%3*(\text{int})(x+y)\%2/4$ ， 设 $x=2.5, a=7, y=4.7$

(2)  $(\text{float})(a+b)/2+(\text{int})x\%(\text{int})y$ ， 设 $a=2, b=3, x=3.5, y=2.5$

先写出运行结果， 然后编制程序验证之。（提交源程序）

3\_4 先写出下列程序的运行结果，然后编制程序验证之。

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

```
void main()
```

```
{
```

```
int i,j,m,n;
```

```
i=8;
```

```
j=10;
```

```
m=++i;
```

```
n=j++;
```

```
printf(“%d,%d,%d,%d”,i,j,m,n);
```

```
}
```

3\_5 写出下列赋值表达式运算后**a**的值，然后编制程序验证之。

设原来**a=12**（提交源程序，注意每次计算之前，将**a**的值复原为12）

(1)  $a+=a$       (2)  $a-=a$       (3)  $a*=2+3$       (4)  $a/=a+a$

(5)  $a\%=(n\%=2)$ ， $n$ 的值等于5      (6)  $a+=a-=a*=a$

以下为自我测试题，做完之后自己对答案

### 3\_6.判断下列陈述是对还是错？

- a) 所有运算符具有相同的优先级.
- b) 求模运算符% 只能用于整型数
- c) 运算符<=,>= 和 !=具有相同的优先级.
- d) 在 C中，如果一个数据项为零，它就被认为是逻辑假.
- e) 表达式!(x<=y) 与x>y等效
- f) 一个一元表达式由一个没有运算符的操作数组成.
- g) 结合性被用来决定具有几个表达式的运算哪个优先执行.
- h) 表达式语句用句点“.”结尾
- i) 对于混合表达式运算，隐式类型转换是自动进行的
- j) 强制类型转换可以用于改变表达式
- k) 小括号可以用于改变表达式

**3\_7. 下列哪些表达式是有效的？如果有效，给出表达式的值；否则说明原因.**

a)  $25/3\%2$

b)  $+9/4+5$

c)  $7.5\%3$

d)  $14\%3+7\%2$

e)  $-14\%3$

f)  $15.25+-5.0$

g)  $(5/3)*3+5\%3$

h)  $21\%(int)4.5$



**3\_8.** 假设有: `int a=10,b=20,c;` 判断下列表达式是对还是错.

- a) 语句 `a=+10`, 是有效的;
- b) 表达式 `a+4/6*6/2` 的值为11;
- c) 表达式 `b+3/2*2/3` 的值为20;
- d) 语句 `a+=b`; 将30赋给a, 将20赋给b;
- e) 语句 `++a++`; 运算的结果为a=12;
- f) 语句 `a/=b`; 将值0.5赋给a ;

**3\_9.** 找出下列运算式中不必要的小括号, 给出其精炼的表达式

- `((x-(y/5)+z)%8)+25`
- `((x-y)*p)+q`
- `(m*n)+(-x/y)`
- `x/(3*y)`



# 答案

## 3\_1

	Error	Correct
Sentence1:	;	(no semicolon sign)
Sentence2:	void mian{}	void main()
Sentence5:	PRINTF	printf
Sentence6:	d%	%d
Sentence7:	%D	%d
Sentence8:	实型数用整型格式输出	%f
Sentence9:	整型数用实型格式输出	%d
Sentence10 :	两个整数相除得到整数	x=3.0/2;
Sentence11 :	实数的小数部分被截尾，精度降低	

## 3\_6.

a) False	b) True	c) True	d) True
e) True	f) False	g) True	h) False
i) True	j) True	k) True	

3\_7.

- a) 0                      b) 7  
c) 无效, 实数不能用%运算                      d) 3                      e) -2  
f) 无效, -5.0 应该有小括号                      g) 5                      h) 1

3\_8.

- a) True   b) False                      c) True  
d) False                      e) True   f) False

3\_9.

- a)  $(x-y/5+z)\%8+25$                       b)  $(x-y)*p+q$   
c)  $m*n-x/y$                       d)  $x/(3*y)$

# 模拟考试题

## 选择题

1. 已知: `char a; int b; float c; double d;`

执行语句“`c=a+b+c+d;`”后, 变量c的数据类型是\_\_。

A) `int` B) `char` C) `float` D) `double`

2. 温度华氏和摄氏的关系是:  $C = 5/9 (F - 32)$ 。已知: `float C, F;` 由华氏求摄氏的正确的赋值表达式是\_\_\_\_\_。

A) `C=5/9(F-32)` B) `C=5*(F-32)/9`

C) `C=5/9*(F-32)` D) 三个表达式都正确

3. 逗号表达式“`(a=3*5,a*4),a+15`”的值是\_\_\_\_\_。

A) 15 B) 60 C) 30 D) 不确定

4. 如果`int a=1,b=2,c=3,d=4;` 则条件表达式“`a<b?a:c<d?c:d`”的值是\_\_\_\_\_。

A) 1 B) 2 C) 3 D) 4

5. 为求出`s=10!`的值, 则变量s的类型应当为\_\_。

A) `int` B) `unsigned` C) `long` D) 以上三种类型均可

# 模拟考试题

- ❖ 6. 已知int x=1,y; 执行下述语句后变量x的值是\_\_\_\_\_。
- ❖ y=++x>5&&++x<10;
- ❖ A) 1 B) 2 C) 3 D) 4
- ❖ 7. 为判断字符变量c的值不是数字也不是字母时, 应采用下述表达式\_\_\_\_\_。
- ❖ A)  $c \leq 48 \parallel c \geq 57 \&\& c \leq 65 \parallel c \geq 90 \&\& c \leq 97 \parallel c \geq 122$
- ❖ B)  $!(c \leq 48 \parallel c \geq 57 \&\& c \leq 65 \parallel c \geq 90 \&\& c \leq 97 \parallel c \geq 122)$
- ❖ C)  $c \geq 48 \&\& c \leq 57 \parallel c \geq 65 \&\& c \leq 90 \parallel c \geq 97 \&\& c \leq 122$
- ❖ D)  $!(c \geq 48 \&\& c \leq 57 \parallel c \geq 65 \&\& c \leq 90 \parallel c \geq 97 \&\& c \leq 122)$
- ❖ 8. 已知 int a=3,b=2,c=1;
- ❖ 则表达式“a/b/c的值是\_\_\_\_\_。
- ❖ A) 1.5 B) 1 C) 0 D) 错误的表达式

# 模拟考试题

9. 已知 `int x=1,y=1,z=1;`

表达式"`x+++y+++z++`"的值是\_\_\_\_\_。

A) 3 B) 4 C) 5 D) 表达式错误

10. 已知`int x=5,y=5,z=5;` 执行语句`x%=y+z;`

后, `x`的值是\_\_\_\_\_。

A) 0 B) 1 C) 5 D) 6

11. 使用语句`scanf("x=%f,y=%f",&x,&y);`

输入变量`x`、`y`的值（□代表空格），正确的输入是\_\_\_\_\_。

A) 1.25,2.4 B) 1.25□2.4 C) `x=1.25,y=2.4` D) `x=1.25□y=2.4`

12. 已知`int x=(1,2,3,4);` 变量`x`的值是\_\_\_\_\_。

A) 1 B) 2 C) 3 D) 4

13. 与条件表达式"`(n)?(c++):(c--)`"中的表达式（`n`）等价的表达式是\_\_\_\_\_。

A) `(n==0)` B) `(n==1)` C) `(n!=0)` D) `(n!=1)`

# 模拟考试题

## ❖ 填空题

1. 若有以下定义：int m=5,y=2;则计算表达式 $y+=y-=m*=y$ 后的y值是\_\_。
2. 在C语言中，一个int型数据在内存中占2个字节，则int型数据的取值范围为\_\_。
3. 已知字母a 的ASCII码为十进制数97，且设ch为字符型变量则表达式 $ch='a'+8-'3'$ 的值为\_\_。
4. 假设所有变量均为整型，则表达式 $(a=2,b=5,a++,b++,a+b)$ 的值为\_\_。
5. 若s是int型变量，且 $s=6$ ，则表达式 $s\%2+(s+1)\%2$ 的值为\_\_。
6. 若a是int型变量，则下面表达式 $(a=4*5,a*2),a+6$ 的值为\_\_。
7. 若x和a均是int型变量，则计算表达式 $x=(a=4,6*2)$ 后的x值为\_\_，计算表达式 $x=a=4,6*2$ 后的x 值为\_\_。
8. 若a是int型变量，则计算表达式 $a=25/3\%3$ 后a的值为\_\_。
9. 若x和n均是int型变量，且x和n的初值均为5，则计算表达式 $x+=n++$ 后x的值为\_\_，n的值\_\_。



# 模拟考试题

10. 若有定义： `int x=3,y=2;float a=2.5,b=3.5;` 则表达式 `(x+y)%2+(int)a/(int)b` 的值为\_\_\_\_\_。
11. 已知字母a的ASCII码为十进制数97， 且设ch为字符型变量， 则表达式 `ch='a'+'8'-'3'` 的值为\_\_\_\_\_。
12. 若有定义： `int e=1,f=4,g=2;float m=10.5 n=4.0 k;` 则执行赋值表达式 `k=(e+f)/g+sqrt((double)n)*1.2/g+m` 后k的值是\_\_\_\_\_。
13. 表达式 `8/4 *(int)2.5/(int)(1.25*(3.7+2.3))` 值的数据类型为\_\_\_\_\_。
14. 假设m是一个三位数， 从左到右用a,b,c表示m各位的数字， 则用m表示从左到右的数字是bac的三位数的表达式是\_\_\_\_\_。

# 模拟考试题答案

## ❖ 选择题

1.C    2.B    3.C    4.A    5.C    6.B    7.D    8.B    9.A  
10.C    11.C    12.D    13.C

## ❖ 填空题

1.-16    2.-32768~+32767    3.'f'或 102    4.9    5. 1  
6.26    7.12,4    8.2    9.10,6    10.1    11.102  
12.13.700000    13.整型（或int型）  
14.  $(m/10) \% 10 * 100 + m/100 * 10 + m \% 10$



# **第四章 最简单的C程序设计**

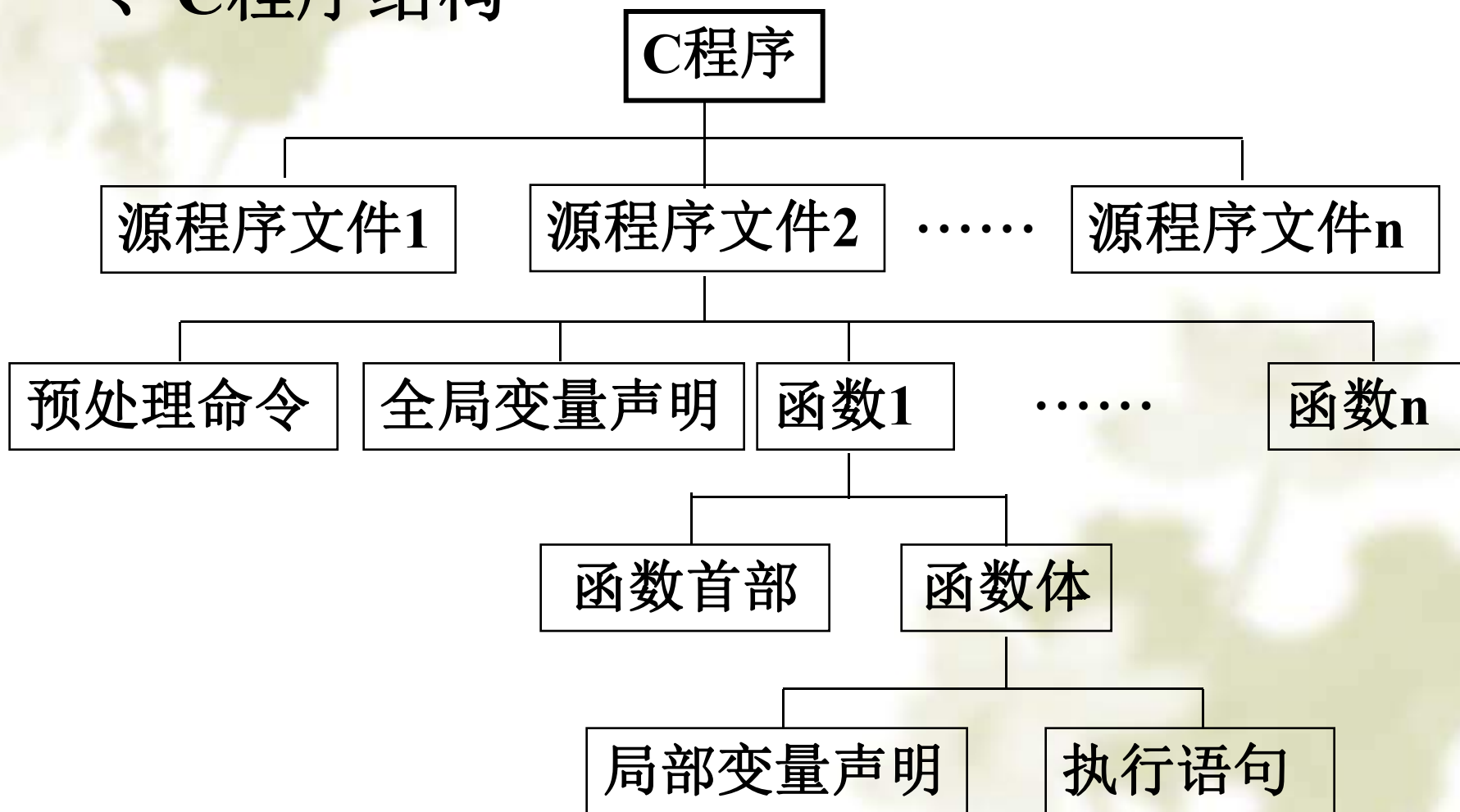
## **The Simplest C Program Design**

### **顺序程序设计**

#### **The Sequence Program Design**

# 4.1 C语句概述(Description of C sentence)

## 一、C程序结构



## 二、C语句概述

### C语句分类: (Assortment)

- 1.控制语句(**Control sentences**)
- 2.函数调用语句(**function Call sentences**)
- 3.表达式语句(**Expression sentences**)
- 4.空语句(**Empty sentences**)
- 5.复合语句(**Compound sentences**)

一条完整的C语句必须以“;”结束。“;”是一个语句不可缺少的一部分。

# 1. 控制语句(Control sentences)

功能：完成一定的控制功能

C 有9种控制语句：

(1)条件语句（Condition sentence）

if(表达式)

语句1 或者 {语句块1}

else

语句2 或者 {语句块2}

“语句块”又称为复合语句，是由花括号将若干条语句结合成的一个整体。

## (2)for循环语句 (Loop sentence)

for(表达式1;表达式2;表达式3)

语句 或者 {语句块}

## (3)当型循环语句 (while loop)

while( 表达式)

语句 或者 {语句块}

## (4)直到型循环语句 (until 1 loop )

do

语句 或者 {语句块}

while( 表达式);

- (5) continue                    (结束本次循环语句)
- (6) switch                    (多分支选择语句)
- (7) break                    (中止执行switch或循环语句)
- (8) goto                    (转向语句)
- (9) return                    (从函数返回数值语句)

## 2. 函数调用语句(Calling function )

由一次函数调用加一个“;”构成。

如: printf(“Hello World!”);

函数调用

分号

y=pow(x-3,2);

y=sin(x);

### 3. 表达式语句(Expression sentences)

- 表达式：用运算符和括号将运算对象(操作对象)连接起来的、符合C语法规则的式子，称为表达式。
- 表达式语句：由表达式加上“;”构成。
- 表达式能构成语句是C语言的一个重要特色。C程序中的大多数语句（包括程序调用语句）是表达式语句，C语言又称为“表达式语言”。

如：  $x+4*y$ ;      算术表达式语句

$4*7, 4+8$ ;      逗号表达式语句

$x=4*7$ ;      赋值表达式语句(赋值语句)



## 4. 空语句(Empty sentences)

由一个 “;”构成。即: ;

作用: ①程序转向点;

② 什么也不做的循环体。

```
while(i<=11) ;
```

## 5. 复合语句(Compound sentences, 语句块)

用 { } 括起的一组语句。

如: if (x1>x2)

```
{  
    y=x1*x1+x2 ;  
    printf("y=%d",y);  
}
```

不能加 “;”

不能省略



## 4.2 赋值语句(Assignment Sentences)

- (1) C语言中的赋值号 “=”是一个运算符。
- (2) 赋值表达式和赋值语句不同，赋值表达式可以被包含在其它表达式之中。

如： if ((a=b)>0) t=a;

它等于：

a=b;

if (a>0) t=a;

如果写成： if ((a=b;)>0) t=a;

错了。if的条件中需要(测试)表达式，而不是语句

## 4.3 数据输入输出的概念及在 C 语言中的实现(Data Input and Output)

### 一、数据输入输出的概念

计算机的输入输出是以计算机主机为主体而言。

输入：从外部通过输入设备（如键盘、磁盘、光盘、扫描仪等）向计算机输入数据。

输出：从计算机向外部设备（如显示屏、打印机、磁盘等）输出数据。

## 二、数据输入输出在 C 语言中的实现

### 1. 调用输入输出函数实现输入输出操作

C 语言本身不提供输入输出语句，输入和输出操作是通过调用 C 语言函数库中输入输出函数来实现的。

如： putchar 函数——输出字符(Character output)

getchar 函数——输入字符(Character input)

printf 函数 ——格式输出(Data output with format)

scanf 函数 ——格式输入(Data input with format)

puts 函数 ——输出字符串(Strings output)

gets 函数 ——输入字符串(Strings input)

1) C语言库函数(stdio.h, string.h, math.h等)已被编译成目标文件（二进制形式）。

2) 源程序必须先**编译**成目标文件。

源程序中的输入输出函数等库函数，在编译时并不被翻译成目标指令。

3) 生成好的目标文件（.obj）必须与系统的函数库（以及其他的库）目标文件进行**连接**，形成可执行文件（.exe）。

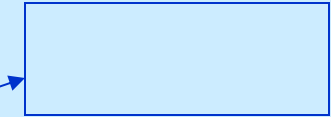
4) **运行**EXE文件时，调用已被连接的函数库中的系统函数。

系统函数库

如： **stdio.h**

```
main()  
{ int a,b;  
  a=10;  
  b=20;  
  printf("a=%d,b=%d",a,b);  
}
```

printf()的目标代码



.....

....

## 2. 文件开头应有“#include”命令

在调用标准输入输出库函数时，文件开头应有预编译命令：

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

或 #include “stdio.h”

其中，stdio.h是“头文件”，包含了与用到的标准输入输出函数有关的信息。stdio.h是standard input& output library head files的缩写。

其他还有

数学函数库“math.h”

字符串函数库“string.h”等。

## 4.4 字符数据的输入与输出(The Input and Output of Character Data)

必须有预编译命令：`#include <stdio.h>`

### 一、`putchar` 函数（字符输出函数）

一般格式：`putchar(c)`

功能：向终端设备输出一个字符。

说明：其中`c`可以是字符型常量、字符型变量、整型常量、整型变量表达式。



例:

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

```
main()
```

```
{
```

```
    char a,b,c;
```

```
    a='B'; b='O'; c='Y';
```

```
    putchar(a);
```

```
    putchar(b);
```

```
    putchar(c);
```

```
    putchar("\n");
```

```
}
```

输出结果: **BOY'**



## 二、getchar函数(字符输入函数)

一般格式：**getchar()** ——没有参数

功能：从终端输入一个字符。

例：**#include "stdio.h"**

**main()**

**{ char c;**

**c=getchar();**

**putchar(c);**

**}**

**a**     ↙

**a**

## 4.5 格式输入输出函数

### (Input and Output Function with Format)

格式字符	用于scanf的输入说明
%c	单个字符
%d	十进制整数
%e	单精度实数
%f	单精度实数
%lf	双精度实数
%g	单精度实数
%u	无符号整数
%i	十进制、十六进制或八进制整数
%o, %x	八进制、十六进制整数
%s	字符串

格式字符	用于printf的输出说明
%c	单个字符
%d	十进制整数
%e	单精度实数
%f	单精度实数
%lf	双精度实数
%g	单精度实数
%i	十进制整数
%o	八进制整数，无前导的0
%s	字符串
%u	无符号整数
%x	十六进制整数，无前导的0x

## 4.5.1 printf 函数（格式输出函数）

功能：向终端输出若干个任意类型的数据。

（一）printf函数的一般格式

**printf（格式控制，输出表列）**

举例：

```
main( )
```

```
{ int a=3,b=4;
```

格式控制

输出表列

```
printf(“a=%d b=%d sum=%d\n”, a, b,a+b );  
}
```

普通字符

格式字符

转义字符

输出结果：

a=3 b=4 sum=7

```
printf(“a=%d b=%d max=%d\n”, a, b,max(a,b) );
```

## printf（格式控制，输出表列）

说明：

“输出表列”是需要输出的一些数据(变量、表达式)。

“格式控制”是用双引号括起来的字符

也称“转换控制字符串”，它包括三种信息：

- 1.普通字符，即需要原样输出的字符。
- 2.格式说明，由“%”和格式字符组成,如%d,%f等。
- 3.转义字符，将\后面的字符转换成另外的意义（P48表），如\n,\t等。

## (二) 格式字符(formatted Character)

对不同类型的数据用不同的格式字符。

**1. d格式符**—— 用来输出十进制整数。

(output the **d**ecimal integer)

(1)%d : 按整型数据的实际长度输出。

如: `int a=123;`

`printf(“%d”,a);`

结果: **123**



(2)%md: m为指定的输出字段的宽度。

若数据位数小于m, 则左端补以空格; 若大于m, 则按实际位数输出。

如: `int a=123, b=12345;`

`printf("%4d,%4d",a,b);`

结果:  `123,12345`

(3)%mld: 输出长整型数据。

m是对长整型数据指定字段宽度。

如: `long a=1234567;`

`printf("%ld,%8ld,%5ld",a,a,a);`

结果: `1234567, 1234567,1234567`

## 2. f格式符——以浮点数形式输出实数。

(output the **f**loating point real number)

(1) %f：不指定字段宽度,由系统自动指定,  
使整数部分全部如数输出,并输出6位小数。

注意float型和double型数据的有效位数。

如: float x=123456.123;

```
printf(“%f”,x);
```

输出结果: 123456.125000



```
main()
```

```
{ double x,y;
```

```
  x=1111111111111111.1111111;
```

```
  y=1111111111111111.1111111;
```

```
  printf("%f",x+y);
```

```
}
```

```
2222222222222222.220000
```

(2) `%m.nf` : 指定输出的数据共占m列,其中有n位小数。若数据长度小于m, 则左端补空格。

(3) `%-m.nf` : 与`%m.nf` 基本相同, 只是使输出的数值向左端靠, 右端补空格。

如: `float f=123.456;`

`printf(“%10.2f,%-10.2f”,f,f);`

输出结果: `123.46,123.46`

(4) `%m.nf` : 若数据长度大于`m`, 或者`m < n`, 则输出具有`n`位小数的原数据。

如: `float f=123.456;`

`printf(“%5.2f,%0.3f”,f,f);`

输出结果: 123.46,123.456

### 3. e格式——以指数形式输出实数

(output the real number in **e**xponent)

(1) %e

如: `printf("%e",123.456);`

输出结果: 1.234560e+002

6位

3位

一般共占13列宽度

(2) %m.ne和 %-m.ne: m、n和“-”字符含义与前相同。此处n指数据的数字部分(又称尾数)的小数位数。

## 4. g格式符——用来输出实数


它根据数值的大小,自动选f格式或e格式(选择输出时占宽度较小的一种), 且不输出无意义的零。

(output the real number in float point or exponent)

如: `f=123.468;`

```
printf(“%f, %e, %g”, f, f, f);
```

输出如下:

**123.468000** , **1.234680e+002** , **123.468** 

10列                      13列                      10列



## 5. o格式符——以8进制数形式输出整数

输出的数值不带符号,即将符号也一起作为八进制数的一部分输出。

如: `int a=-1;`

1 111 111 111 111 111

`printf ( “%d,%o”,a,a) ;`

结果: **-1, 177777**

o格式符同d格式符一样, 可以输出长整型数据, 也可以指定字段宽度。

**6. x格式符**——以16进制数形式输出整数  
同样不会出现负的十六进制数。

**7. u格式符**——以十进制形式输出**unsigned**型数据

如: `main()`

```
{ unsigned int a=65535;  
  int b=-2;  
  printf("a=%d,%o,%x,%u\n",a,a,a,a);  
  printf("b=%d,%o,%x,%u\n",b,b,b,b);}
```

输出结果:     **a=-1,177777,ffff,65535**

**b=-2,177776,fffe,65534**

## 8. C格式符——用来输出一个字符

一个整数,只要它的值在0~255范围内,也可以用字符形式输出,在输出前,将该整数转换成相应的ASCII字符;反之,一个字符数据也可以用整数形式输出。

如: `main()`

```
{ char c='a'; int i=97;  
  printf("%c,%d\n",c,c);  
  printf("%c,%d\n",i,i);}
```

输出结果:    **a,97**  
                 **a,97**

## 9. s格式符——用来输出一个字符串

(1) %s : 按原长度输出字符串。

如: `printf(“%s”,“CHINA”)`

输出结果: CHINA

(2) %ms : 输出的字符串占m列, 如字符串本身长度大于m, 则突破m的限制, 将字符串全部输出; 若串长小于m, 则左补空格。

(3) %-ms : 如果串长小于m, 则在m列范围内, 字符串向左靠, 右补空格。

(4) `%m.ns` : 输出占m列，但只取字符串中左端n个字符。输出在列的右侧，左补空格。如果 $n > m$ ，则自动取n值，即保证n个字符正常输出。

(5) `%-m.ns` , 其中m、n含义同上，n个字符输出在m列范围的左侧，右补空格。

如： `printf(“%3s,%7.2s,%.4s,%-5.3s\n”,  
“CHINA”, “CHINA”, “CHINA”, “CHINA”);`

输出结果： CHINA, CH, CHIN, CHI

## 4.5.2 scanf 函数（格式输入函数）

### （一）一般形式

**scanf（格式控制，地址表列）**

**main()**

**{ int a,b,c;**

**scanf(“%d,%d”,&a,&b);**

**printf(“a=%d,b=%d”,a,b);**

**输入： 10,15**

**输出： a=10,b=15**

变量名	值	地址
-----	---	----

a	10	2008
---	----	------

b	15	2010
---	----	------

c		2012
---	--	------

## (二) 格式说明

与printf函数中的格式说明相似，以%开始，以一个格式字符结束（表4-3），中间可以插入附加的字符（表4-4）。

如： `int a,b,c;`

`scanf(“%d%d%d”,&a,&b,&c);`

输入形式： 10 17 181

---

`int a,b,c;`

`scanf(“%d,%d,%d\n”,&a,&b,&c);`

输入形式： 10,11,12

---



**scanf(“%3d%3d%c”,&a,&b,&ch);**

输入形式: 123456abc

结果: **a=123 b=456 ch='a'**

---

**scanf(“%d%d%c”,&a,&b,&ch);**

输入形式: 123\_456abc

结果: **a=123 b=456 ch='a'**

有错吗？

```
main()
```

```
{ int x,y;
```

```
    char ch;
```

```
    scanf("%5d,%3d%c",x,y,ch);
```

```
}
```

```
main()
```

```
{ int x,y;
```

```
    char ch;
```

```
scanf("%5d,%3d%c",&x,&y,&ch);
```

```
}
```

输入数据：   345,567a

# 说明:

1. %后的“\*”附加说明符，用来表示跳过它相应的数据。

如: `scanf("%2d,%*3d,%2d",&a,&b);`

输入: 12, 456, 67

结果: **a=12 b=67**

---

2. 输入数据时不能规定精度。

✗ `scanf("%7.2f",&a);`

---

3. 输入的字符、数值要与格式一致

`scanf("a=%d,b=%d",&a,&b);`

输入: a=12,b=13

---

4、`scanf("%c%c%c",&a,&b,&c);`

输入: a **\_** b **\_** c

空格和转义字符都作为有效字符输入

当要求

输入数据： a=10 b=20 x=30 y=40 ‘a’ ‘b’

输出数据： 10,20,30.000000,40.000000,a,b时

不要采用如下输入方式：

main()

```
{ int a,b; float x,y; char c1,c2;  
  scanf("a=%d b=%d",&a,&b);  
  scanf(" x=%f y=%e",&x,&y);  
  scanf(" c1=%c c2=%c",&c1,&c2);  
  printf("%d,%d,%f,%f,%c,%c", a,b,x,y,c1,c2);  
}
```

要采用如下输入方式:

```
main()
```

```
{ int a,b; float x,y; char c1,c2;  
  printf("a= b=\n");  
  scanf("%d %d",&a,&b);  
  printf("x= y=\n");  
  scanf("%f %f",&x,&y);  
  printf("c1= c2=\n");  
  scanf("%c %c",&c1,&c2);  
  
  printf("%d,%d,%f,%f,%c,%c",a,b,x,y,c1,c2);  
}
```

输入数据：

a= b=

10 20

x= y=

30 40

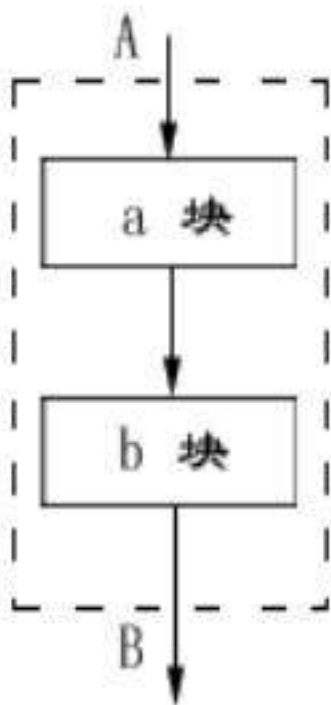
c1= c2=

ab

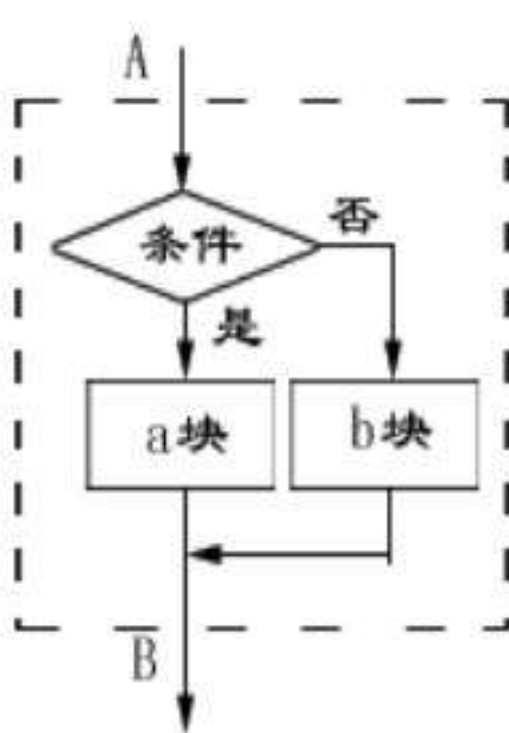
输出数据： 10,20,30.000000,40.000000,a,b

## 4.6 顺序结构程序设计举例

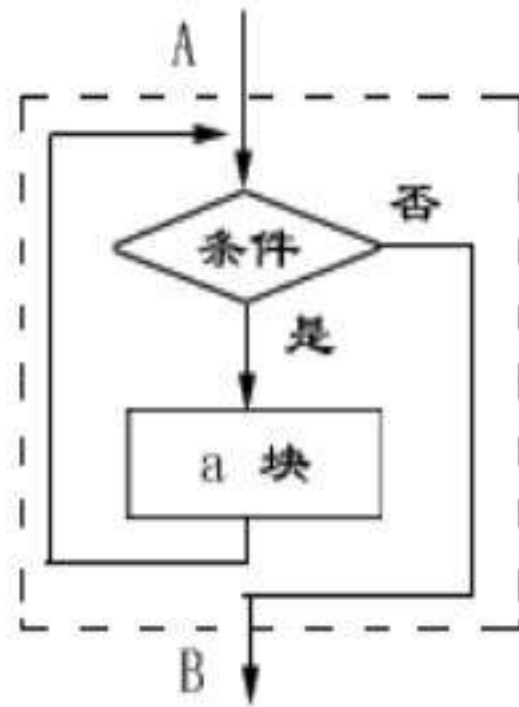
特点：程序按语句从上到下的排列顺序依次执行，每条语句必须执行且只能执行一次，没有执行不到或执行多次的语句。



(顺序结构)



(选择结构)



(循环结构)



例如：已知圆柱体半径6米，高7米，求体积。



```
#define PI 3.14159
main( )
{
    float r , h , s , v ;
    r=6 ;
    h=7 ;
    s=PI*r*r ;
    v=s*h ;
    printf("V=%f\n", v);
}
```

## 例4.10 输入三角形的三条边，求三角形的面积。

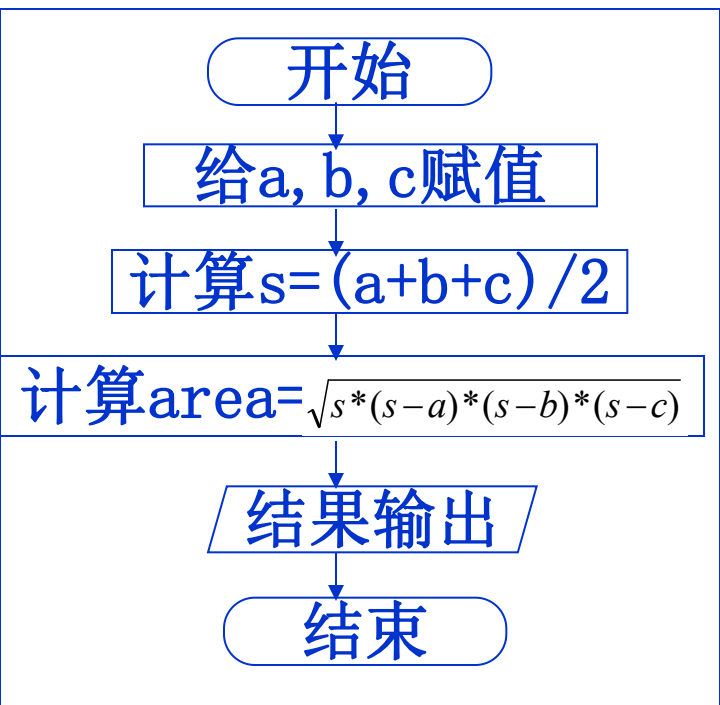
已知：三角形的三条边  $a, b, c$ 。

求解：三角形的面积  $area$ 。

计算公式：设  $s = (a+b+c)/2$

$area =$

$$\sqrt{s * (s - a) * (s - b) * (s - c)}$$



```
#include <math.h>
```

```
void main()
```

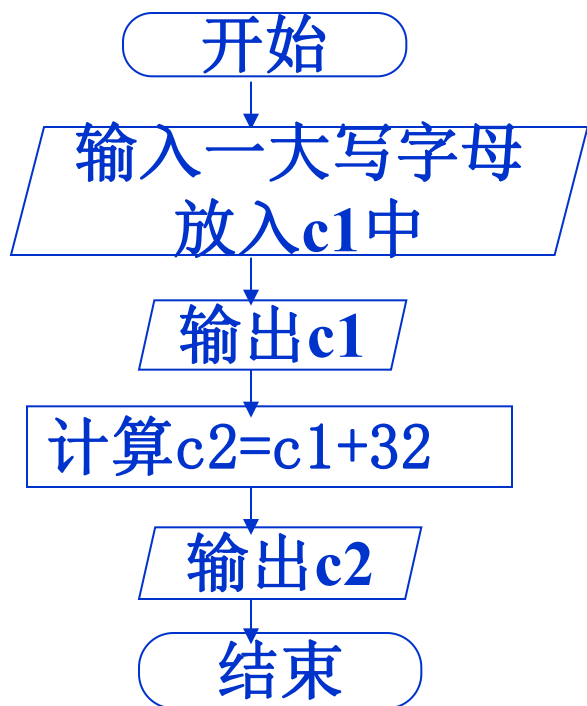
```
{  
    float a,b,c,s,area;  
    scanf("%f%f%f",&a,&b,&c);  
    s=(a+b+c)/2.0;  
    area=sqrt(s*(s-a)*(s-b)*(s-c));  
    printf("a=%f b=%f c=%f s=%f"  
    ,a,b,c,s);  
    printf("area=%f",area);  
}
```

## 例4.11 从键盘输入一个大写字母，要求改用小写字母输出。

已知：字符变量c1为大写字母。

求解：将字符变量c2中放入相应的小写字母。

计算公式： $c2=c1+32$  （小写改成大写字母？）



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

```
main()
```

```
{
```

```
    char c1,c2;
```

```
    c1=getchar(); /*键盘输入1个大写字母*/
```

```
    printf("%c,%d\n",c1,c1);
```

```
    c2=c1+32;
```

```
    printf("%c,%d\n",c2,c2);
```

```
}
```

输入:A✓

输出:A,65  
a,97

## 例4.12 求 $ax^2+bx+c=0$ 的根， $a$ ， $b$ ， $c$ 由键盘输入

```
#include<stdio.h>
#include <math.h>
main()
{
    double a,b,c,disc,x1,x2;
    printf("Input values of a,b,and c\n");
    scanf("%f%f%f",&a,&b,&c);
    disc=b*b-4*a*c;
    if(disc<0)
        printf("\nRoots Are Imaginary\n");
    else{
        x1=(-b+sqrt(disc) )/(2*a);
        x2=(-b-sqrt(disc) )/(2*a);
        printf("x1=%f\nx2=%f",x1,x2);
    }
}
```

运行情况:

1 3 2



x1=-1.000000

x2=-2.000000

1 2 3



Roots Are Imaginary

# 要牢记

- ❖ 在**scanf()**函数的输入表列中，对基本型的变量，要使用地址运算符**&**。
- ❖ 字符串常数要使用一对双引号“ ”。
- ❖ 单个字符常数要使用一对单引号‘ ’。
- ❖ 不要对输入给定任何精度。
- ❖ 当输入或输出变量的类型与格式字符不一致时，会在运行时出现错误。
- ❖ 不要在**scanf**语句的格式字符中使用逗号。
- ❖ 除了语法问题外，**C**不对格式说明的正确性进行检查。

# 本章作业:

一、 做书上其他习题，自己对答案

二、 编程上机操作题,熟悉各种类型的输入输出的方法，能正确使用各种格式转换符:

4\_1 如果想得到以下的输出格式和结果,请写出程序（包括定义变量类型和设计输出）（`\` 表示1个空格）

`a= \ 3 \ \b= \4 \ \c= \5`

`x=1.200000,y=2.400000,z=-3.600000`

`u= \51274 \ \n= \ \ \128765`

`c1='a' \or \97`

`c2='b' \or \ 98`

**4\_2** 设圆半径**r=1.5**，圆柱高**h=3**，求圆周长、圆面积、圆球表面积、圆球体积、圆柱体积。用**scanf**输入数据，用**printf**输出计算结果，输出时要有文字说明，数据取小数点后**2**位数字。

**4\_3** 输入一个摄氏温度，要求输出华氏温度。公式为

$$F = \frac{9}{5} C + 32$$

输出要有文字说明，取**2**位小数。例如，可以输入摄氏温度**C=0, 50, 100, 200**



以下为自我测试题，做完之后自己对答案

## 4\_4 判断下列说法是真还是假

- a) 头文件<stdio.h>的用途是存储用户编制的程序。
- b) 用于从键盘接受一个字符的标准C函数是**getchar**。
- c) **getchar** 不能用于从键盘接受一行文本。
- d) **scanf** 语句的输入表列可以为一个或多个变量。
- e) 变量构成了**printf**语句格式控制字符串的合法元素。
- f) **scanf** 函数不能用于从键盘读入单个字符。
- g) 如果格式说明符的长度比实际数值的长度大，则该数值将被右对齐打印。
- h) **printf** 语句的打印变量表列可以包含程序调用。
- i) 格式说明%5s 将只打印所给字符串的头五个字符。

## 4\_5 填空

- a) 格式说明\_\_\_\_\_用于短整形数的读和写。
- b) 当使用字符函数时，必须把头文件\_\_\_\_\_包含到程序中。
- c) 当要读一个长双精度数据时，必须使用格式说明\_\_\_\_\_。
- d) 函数\_\_\_\_\_可以用于从屏幕读一个数据，而不用将其赋值给任何变量。
- e) 为使数据左对齐打印，必须在格式说明中使用符号\_\_\_\_\_。

## 4\_6 写出下列**printf** 函数的输出结果

- a) `printf ("%d%c%f",10,'x',1.23);`
- b) `printf ("%2d %c %4.2f",1234,'x',1.23);`
- c) `printf ("%d\t%4.2f",1234,456);`
- d) `printf ("%08.2f",123.4);`
- e) `printf ("%d%d %d",10,20);`

4\_7 指出下列输入语句中的错误，如果有的话。  
假设：

```
int year, count;  
float amount, price;  
char code,city[10];  
double root;
```

- a) scanf ("%c%f%d",city,&price,&year);
- b) scanf ("%s%d",city,amount);
- c) scanf ("%f, %d ,&amount, &year);
- d) scanf (\n"%f",root);
- e) scanf ("%c %d %lf", \*code, &count, Root);

4\_8 给出下列输出语句的输出结果。假设：

```
int count=1275; float price=-235.74;
```

```
char city[10]="Cambridge";
```

- a) `printf ("%d %f", count, price);`
- b) `printf ("%2d\n%f", count, price);`
- c) `printf ("%d %f", price , count);`
- d) `printf ("%10dxxxx%5.2f", count, price);`
- e) `printf ("%s", city);`
- f) `printf ("% -10d % -15s", count, city);`

# 答案

4\_4

- |          |         |         |          |         |
|----------|---------|---------|----------|---------|
| a) False | b) True | c) True | d) True  | e) True |
| f) False | g) True | h) True | i) False |         |

4\_5

- |              |            |        |
|--------------|------------|--------|
| a) %d        | b) stdio.h | c) %lf |
| d) getchar() | e) –       |        |

4\_6

10x1.230000

1234 x 1.23

1234      0.00

“00123.40”

1020 2367460

## 4\_7 改错

- a) scanf ("%s%f%d",city,&price,&year);
- b) scanf ("%s%d",city,&amount);
- c) scanf ("%f, %d",&amount, &year);
- d) scanf ("%f", &root);
- e) scanf ("%c %d %lf", &code, &count, &root);

## 4\_8

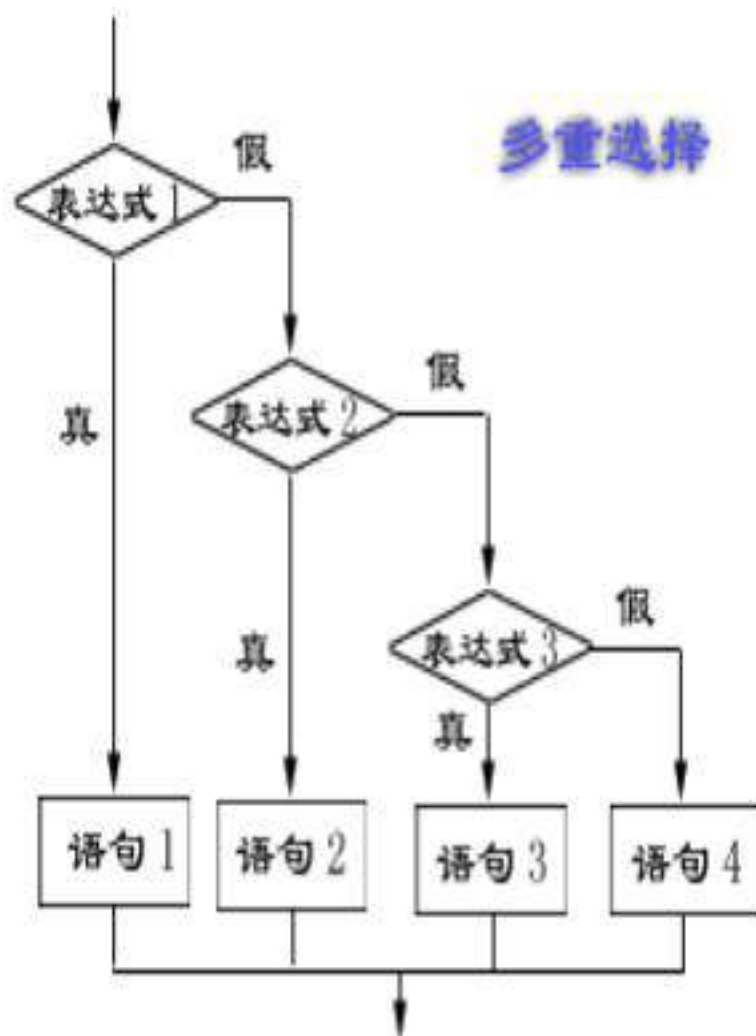
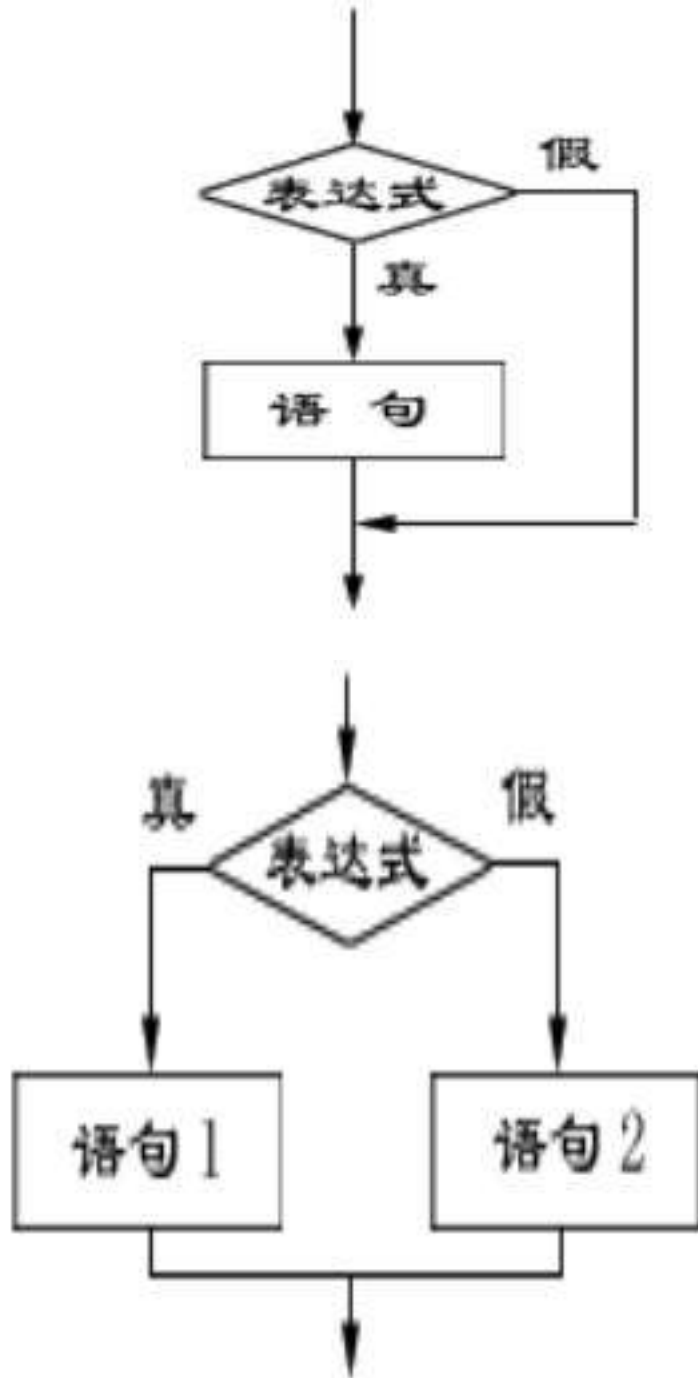
- a) 1275 -235.740005
- b) 1275  
-235.740005
- c) 536870912 0.000000
- d) 1275xxxx-235.74
- e) Cambridge
- f) 1275 Cambridge





# **第5章 选择结构程序设计**

## **Program Design of The Selection Type Structure**



## 5.1 关系运算符和关系表达式

### Relation Operators and Expressions

#### 一、关系运算符及其优先次序

##### 6 种关系运算符:

①  $<$  (小于 **less than**)

②  $\leq$  (小于或等于)

③  $>$  (大于 **larger than**)

④  $\geq$  (大于或等于)

⑤  $==$  (等于 **equal to**)

⑥  $!=$  (不等于 **not equal to**)

} 优先级相同(高)

} 优先级相同(低)

结合性: 自左向右

## 5.1 关系运算符和关系表达式

### 一、关系运算符及其优先次序

与其它运算符优先级的比较：

算术运算符

(高)

关系运算符

赋值运算符

(低)

例如： $c=d>a+b$

等效于： $c=(d>(a+b))$

## 二、关系表达式

### 1. 什么是关系表达式？

如：  $(a > b) > (b < c)$  ,  $'a' < 'b'$  ,  $x > y$  ,  $s + f > d$

### 2. 关系表达式的值

运算结果：逻辑值 1或非零——“真” , 0——“假”

如： 设  $a=3, b=2, c=1$  求下面关系表达式的值：

$c > b$  值为 0

$(a - b) == 1$  值为 1

$b + c < a$  值为 0

如果关系运算对象的类型不同，也要先进行类型转换。

## 5.2 逻辑运算符和逻辑表达式

### 一、逻辑运算符及其优先次序

3种逻辑运算符:

优先级

① **!** (逻辑**非**)

(高)

② **&&** (逻辑**与**)

③ **||** (逻辑**或**)

↓  
(低)

如: **!a**      a为真, 则!**a**为假。

**a||b**      a、b之一为真, 则**a||b**为真。

**a&&b**      a、b 同时为真, 则 **a&&b** 为真。

## 5.2 逻辑运算符和逻辑表达式

### 一、逻辑运算符及其优先次序

与其它运算符的比较:

(高)

例如:  $c = d > a + b \&\& a > b$   
等效于:

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

(低)

1. ( )
2. !, ++, --, -(负号), (类型)
3. 算术 \*, /, %
4. 算术 +, -
6. 关系 <, <=, >, >=
7. 关系 ==, !=
11. 逻辑 &&
12. 逻辑 ||
14. 赋值运算符
15. 逗号运算符



## 二、逻辑表达式

### 1. 什么是逻辑表达式？

用逻辑运算符将运算对象连结起来的式子。

如：  $(a > b) \&\& (b < c)$  、  $!a \parallel a > b$

### 2. 逻辑表达式的值

运算对象： 以数值非0为真，以0为假

运算结果： 逻辑值(只有假为0、真为1两种取值)

如：  $5 > 3 \&\& 2 \parallel 8 < 4$       值为 **1**

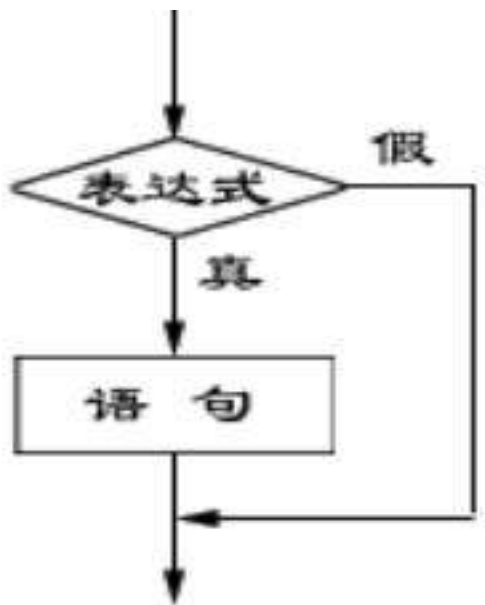
$'b' \&\& 'd'$                       值为 **1**

## 5.3 if语句

### 一、if 语句的三种形式

#### 1. if (表达式) 语句

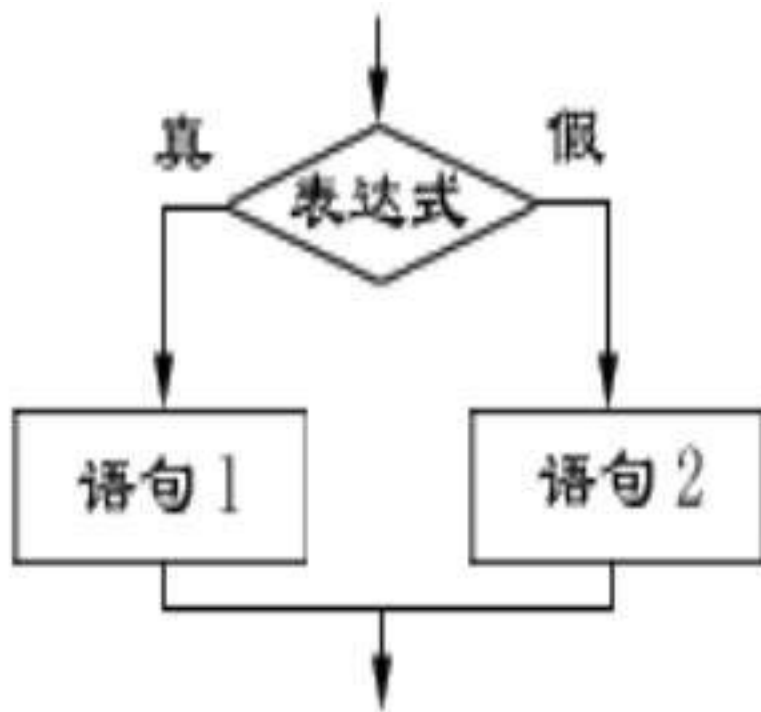
执行过程：当表达式的值为“真” (非零)时，执行语句，否则，不执行语句。



如： `if ( x > y ) z = x;`

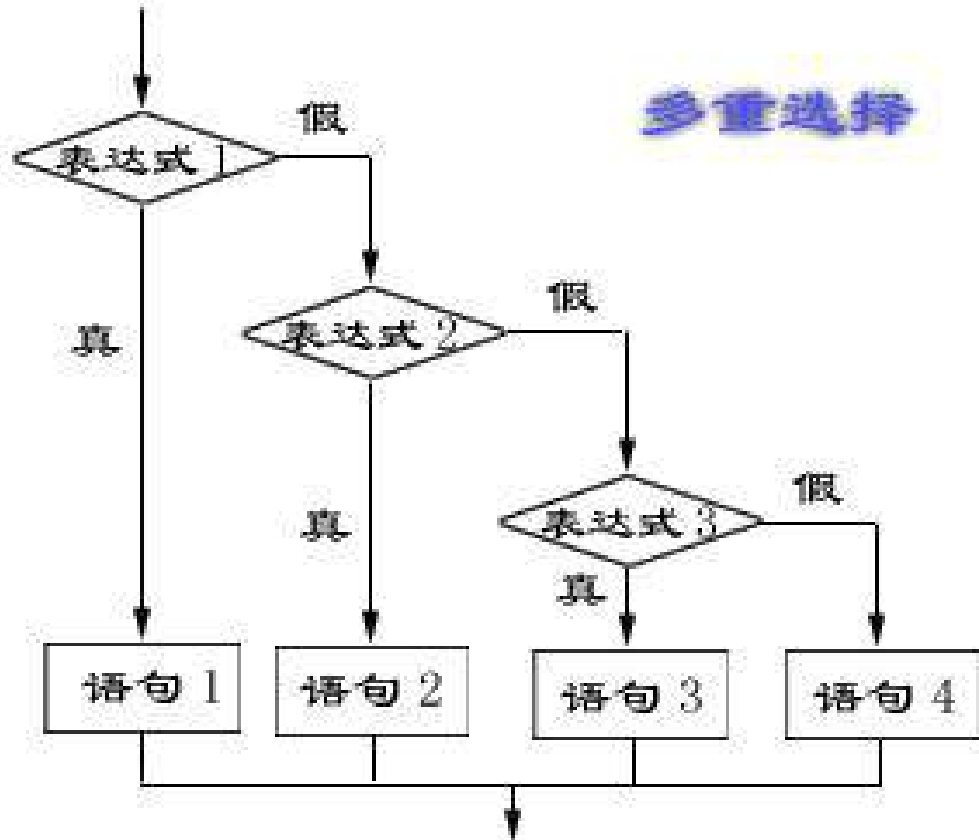
## 2. if (表达式) 语句1 else 语句2

执行过程：当表达式的值为“真”（非零）时，  
执行语句1，否则，执行语句2。



如：if (x>y) z=x ;  
else z=y ;

3. if (表达式1) 语句1  
else if(表达式2) 语句2  
else if(表达式3) 语句3  
:  
else if(表达式n) 语句n  
else 语句n+1



如: if (grade>=85) level= 'A' ;  
else if (grade>=70) level= 'B' ;  
else if (grade>=60) level= 'C' ;  
else level= 'D' ;

## if 语句说明:

(1) if和else if后面的表达式一般为逻辑表达式或关系表达式，也可以是任意数值类型表达式。

该语句是合法的：if(3.0) printf(“ok!”);

(2) if 语句中的“语句”为一个简单语句或复合语句。  
如：下面哪个程序段有错？

```
if (x>=0)
    printf(“X= %d\n”,x);
    printf(“X is positive.\n”);
else
    printf(“X= %d\n”,x);
    printf(“X is negative.”);
```

```
if (x>=0)
{   printf(“X= %d\n”,x);
    printf(“X is positive.\n”);}
else
{   printf(“X= %d\n”,x);
    printf(“X is negative.”);}
```

## 二、if 语句的嵌套

在 if 语句中又包含一个或多个 if 语句，称为 **if 语句的嵌套**。

一般形式：

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

内嵌if

内嵌if

如：

```
if(x>0)
    if(y>0)
        printf("x>0,y>0");
    else
        printf("x>0, y<=0");
else
    if(y>0)
        printf("x<=0, y>0");
    else
        printf("x<=0, y<=0");
```

## 说明:

else与if的匹配原则是：一个else应与其之前距离最近且没有与其它else配对的if 配对。

```
if(n >= 18)
{
    if(n < 60)
        printf("adult");
    else
        printf("minor");
}
```

```
if (n >= 18)
{
    if (n < 60) printf("adult");
}
else
    printf("minor");
```

```
if (n >= 18)
    if (n < 60)
        printf("adult");
    else ;
else
    printf("minor");
```



# 缩 排

```
#include <stdio.h>
main()
{
    缩进 int x;
    scanf("%d",&x);
    if (x>0)
    {
        缩进 printf("X is positive.");
        x=x+1;
    }
    else
        缩进 printf ("X is negative.");
}
```

### 三、用if 语句实现选择结构举例

例： 求  $y = \begin{cases} -1 ; & x < 0 \\ 0 ; & x = 0 \\ 1 ; & x > 0 \end{cases}$  输入x 值， 输出y值。

```
main( )
{
    int x,y;
    scanf("%d",&x);
    if(x<0) y=-1;
    else if(x==0) y=0;
    else if(x>0) y=1;
    printf("x=%d y=%d",x,y);
}
```

例：任意输入三个不相等的数，将其按由小到大的顺序输出。

```
main()
```

```
{
```

```
    float a,b,c,t;
```

```
    scanf("%f%f%f",&a,&b,&c);
```

```
    if(a>b)
```

```
        { t=a;a=b;b=t;} //结果a<b
```

```
    if(a>c)
```

```
        {t=a;a=c;c=t;} //结果a<c
```

```
    if(b>c)
```

```
        { t=b;b=c;c=t;} //结果b<c
```

```
    printf("%5.2f,%5.2f,%5.2f",a,b,c); //a<b<c
```

```
}
```

a=7,b=3

t=a(=7)

a=b(=3)

b=t(=7)

## 注意区别

```
if(season==spring) printf("flower");  
else if(season==summer) printf("hot");  
else if(season==autumn) printf("fruit");  
else printf("snow");
```

```
if(e1) s1;  
else if(e2) s2;  
else s3;
```

```
if(sex=="male") printf("Man");  
if(age>=18) printf("adult");  
if(profession=="armyman") printf("PLA");
```

```
if(e1) s1;  
if(e2) s2;  
if(e3) s3;
```

## 四、条件运算符和条件表达式

### 条件表达式的一般形式:

表达式1 ? 表达式2 : 表达式3    如:  $(a > b) ? a : b$

条件表达式 =  $\begin{cases} \text{表达式2 (表达式1为真)} \\ \text{表达式3 (表达式1为假)} \end{cases}$

C中唯一的三目运算符。目的: 简化 if 语句。

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

若  $a=3, b=8$ , 则  $\text{max}=?$

等效于: `max = (a > b) ? a : b ;`

## 条件运算符的优先级(13级):

算术运算符=>关系运算符=>逻辑运算符(除! )  
(3、4级)                      (6、7级)                      (11、12级)

逗号运算符<=赋值运算符<=条件运算符  
(15级)                      (14级)



如:  $x = a + b > c \ \&\& \ c > d ? a + b : d$

等效于:  $x = (((a + b) > c) \ \&\& \ (c > d)) ? (a + b) : d$

条件运算符的结合方向: 自右至左

如:  $a > b ? a : c > d ? c : d$

等效于:  $a > b ? a : (c > d ? c : d)$

例：把输入字符中的小写字母转换成大写字母并输出，如果是其他字符，则原样输出。

```
#include <stdio.h>
main()
{
    char ch;
    scanf("%c",&ch);
    ch=(ch>='a' &&ch<='z') ? ch-32: ch;
    printf("%c",ch);
}
```

输入数据： **b**

输出结果： **B**



例：输入3个数，把其中最大的数输出

main()

```
{  
    int a,b,c,t,max;  
    scanf("%d%d%d",&a,&b,&c);  
    t=(a>b) ? a : b ;  
    max=(t>c) ? t : c ;  
    printf("max=%d\n",max);  
}
```

或者

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

或者

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

## 5.4 switch 语句

一般形式:

switch (表达式)

{

case 常量表达式1: 语句序列1 [break;]

case 常量表达式2: 语句序列2 [break;]

:

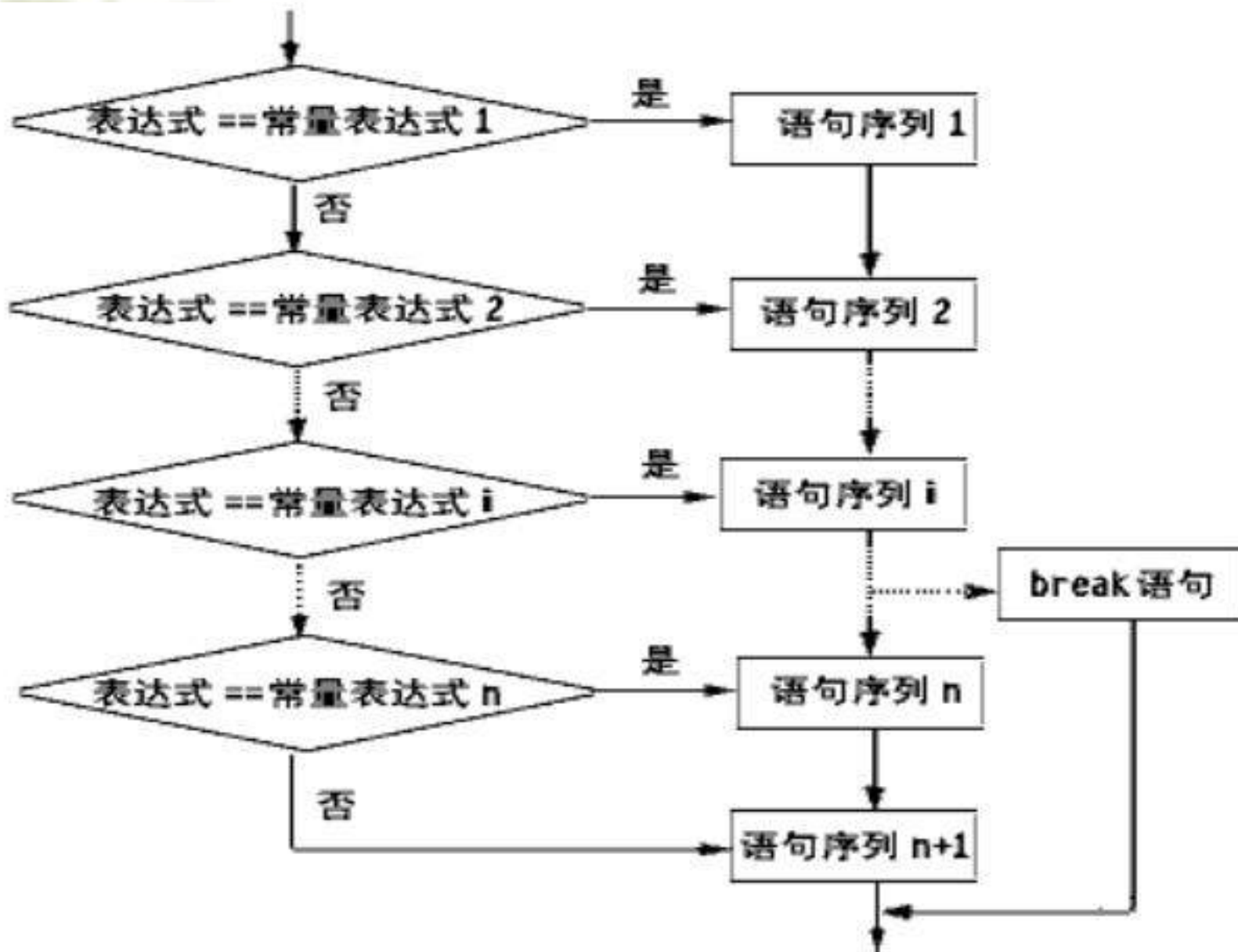
case 常量表达式n: 语句序列n [break;]

[default : 语句序列n+1 ]

}

// default——缺省

## 执行过程:



例：

```
char grade;
```

```
.....
```

```
switch(grade)
```

```
{
```

```
    case 'A' : printf("85~100  ");
```

```
    case 'B' : printf("70~84  ");
```

```
    case 'C' : printf("60~69  ");
```

```
    case 'D' : printf("<60  ");
```

```
    default : printf("error ");
```

```
}
```

若grede= 'A'， 输出结果是什么？

**85~100    70~84    60~69    <60    error**

用break语句处理后的程序段：

```
switch(grade)
{
    case 'A' : printf("85~100 "); break;
    case 'B' : printf("70~84 "); break;
    case 'C' : printf("60~69 "); break;
    case 'D' : printf("<60 "); break;
    default : printf("error ");
}
```

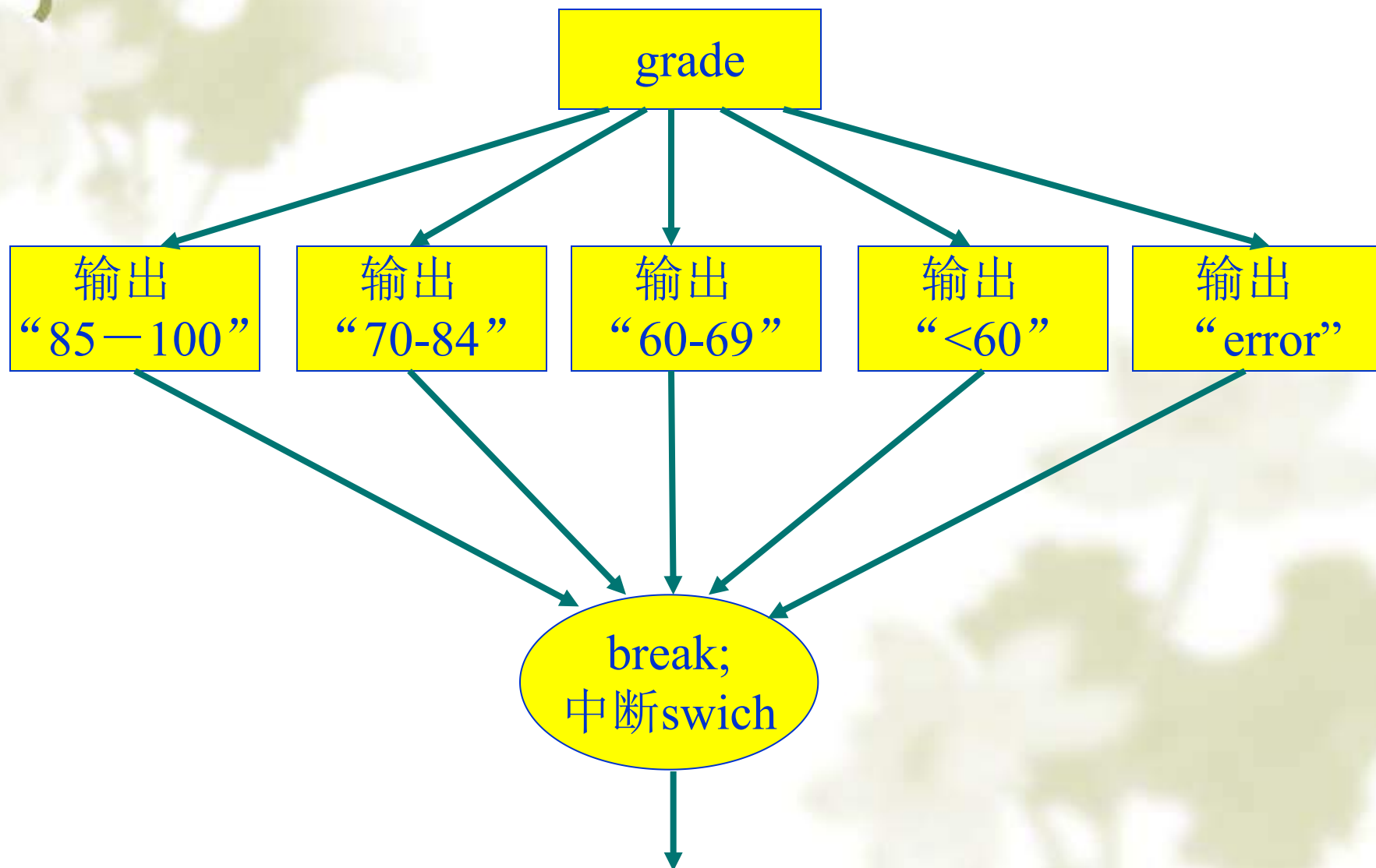
若gread= 'A'，输出结果又是什么？

**85~100**

## 完整程序：

```
main()
{
    char grade;
    printf("Input grade(A,B,C,D):");
    scanf("%c",&grade);
    switch(grade)
    {
        case 'A' : printf("85~100 "); break;
        case 'B' : printf("70~84 "); break;
        case 'C' : printf("60~69 "); break;
        case 'D' : printf("<60 "); break;
        default : printf("error ! ");
    }
}
```

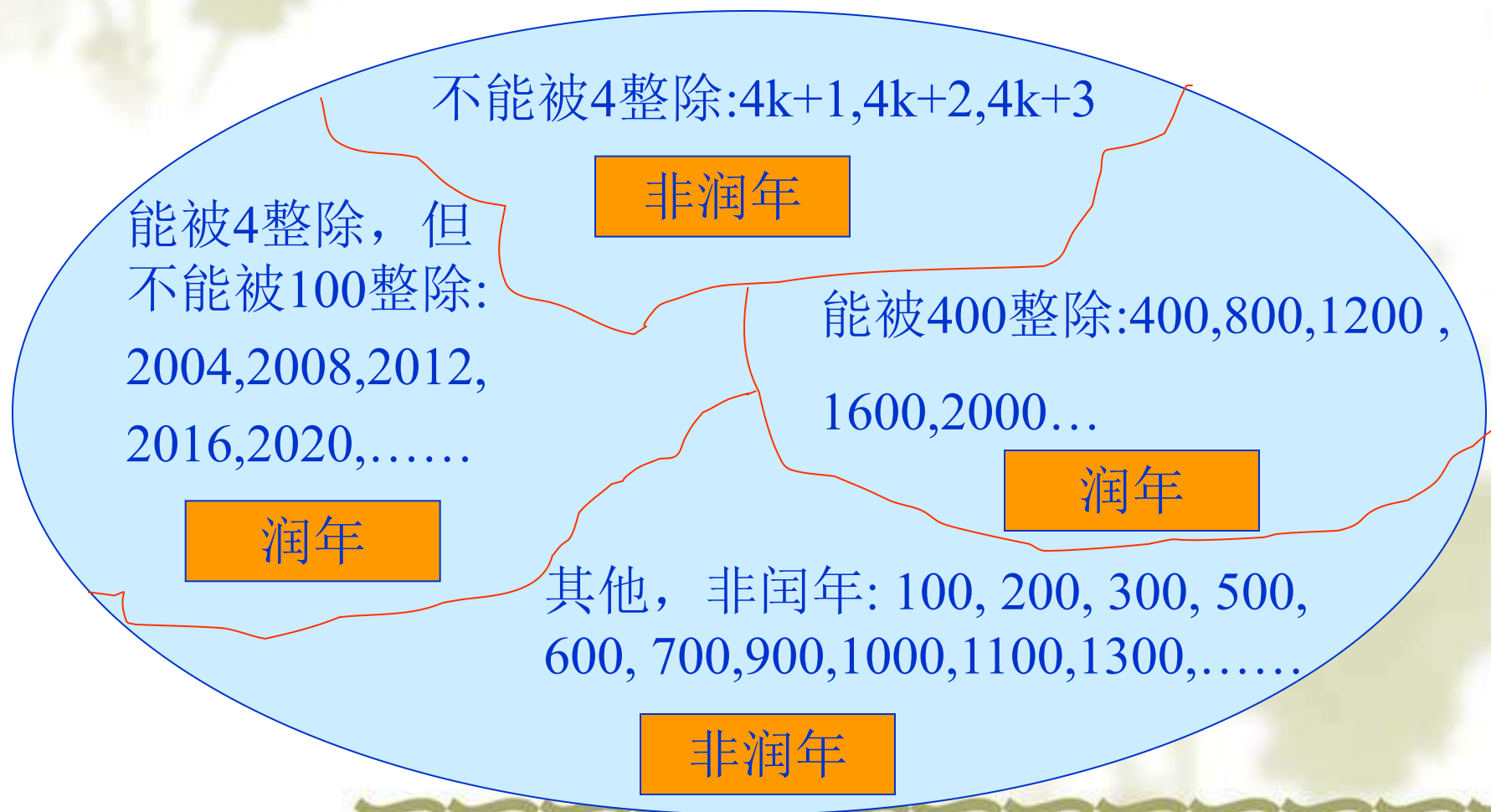
流程图为：





## 5.5 选择结构程序举例

例 1 写程序，判断某一年是否闰年。



设： leap代表闰年信息， leap=1闰年， leap=0 非闰年

编程一（采用复合运算表达式）：

```
main( )
```

```
{
```

```
    int year, leap=0;
```

```
    scanf("%d", &year); /* 输入年份值 */
```

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

```
        leap=1;
```

```
    else leap=0;
```

```
    if(leap==1) printf("%d is leap ", year);
```

```
    else printf("%d is not leap ", year);
```

```
}
```

算术运算符=>关系运算符=>逻辑运算符(除！)

赋值运算符<=条件运算符

## 编程二（采用阶梯if语句）：

```
main( )
```

```
{
```

```
    int year, leap=0;
```

```
    scanf("%d", &year); /* 输入年份值 */
```

```
    if(year%4 != 0)
```

```
        leap=0;
```

```
    else if(year%100 != 0)
```

```
        leap=1;
```

```
    else if(year%400 == 0)
```

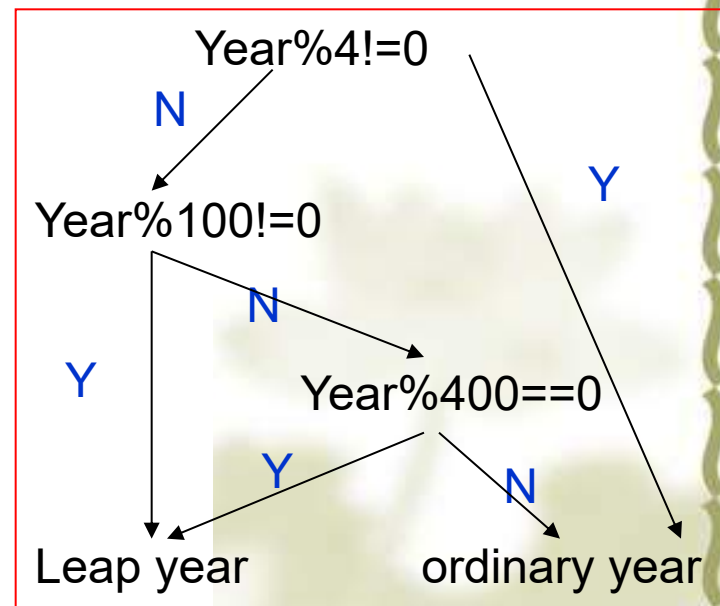
```
        leap=1;
```

```
    else leap=0;
```

```
    if(leap==1) printf("%d is leap ", year);
```

```
    else printf("%d is not leap ", year);
```

```
}
```



原程序段

```
If (T>=180&&M1>=60&&M2>=60 || T>=200)
    printf(" Admitted\n");
else
    printf(" Not admitted\n");
```

不使用采用复合运算表达式，重写上述程序段。

修改一

```
If (T>=200)
    printf(" Admitted\n");
else if(T>=180)
    if(M1>=60)
        if(M2>=60)
            printf(" Admitted\n");
        else
            printf(" Not admitted\n");
    else
        printf(" Not admitted\n");
else
    printf(" Not admitted\n");
```

修改二

```
If (T>=200)
    printf(" Admitted\n");
else if(T<180)
    printf(" Not admitted\n");
else if(M1<60) //M1<60
    printf(" Not admitted\n");
else if(M2<60) //M2<60
    printf(" Not admitted\n");
else printf(" Admitted\n");
//T>=180, M1>=60, M2>=60
```

例5.7 已知基本运费、货重，以及路程(s)和折扣的关系标准如下：

$s < 250\text{km}$	没有折扣
$250 \leq s < 500$	%2折扣
$500 \leq s < 1000$	5%折扣
$1000 \leq s < 2000$	8%折扣
$2000 \leq s < 3000$	10%折扣
$s \geq 3000$	15%折扣

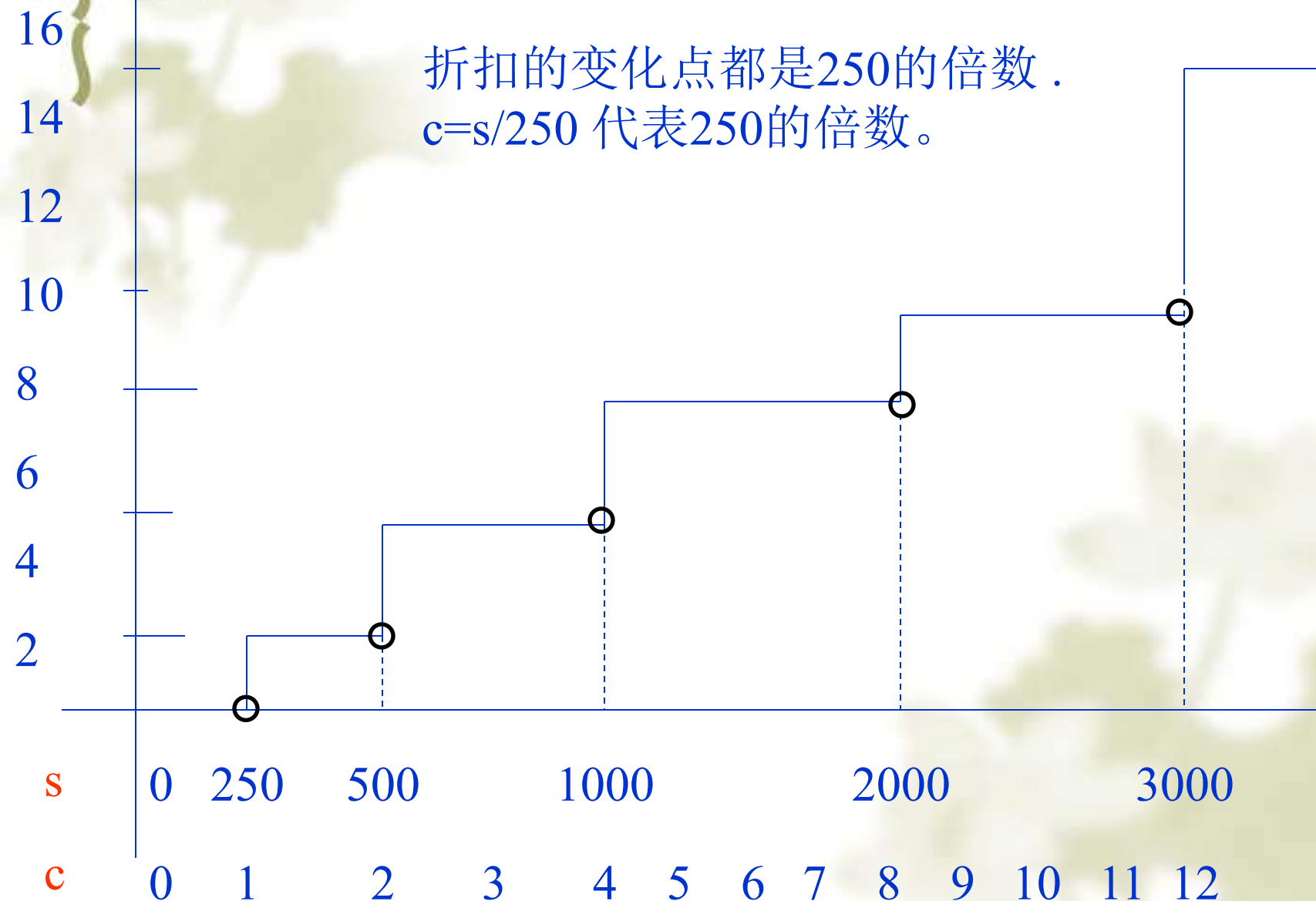
编程序，计算运费f。

设基本运费p、货物重量w、距离s和折扣d。则总运费的计算公式为： $f = p * w * s * (1 - d)$

算法：先按距离的250倍数进行分类，求每类的折扣，再按公式计算。

总运费的计算公式为:  $f=p*w*s*(1-d)$

折扣的变化点都是250的倍数。  
 $c=s/250$  代表250的倍数。



程序为:

main( )

{ int c,s;

float p, w, d , f ;

scanf(“%f , %f , %d”, &p, &w, &s);

if(s>=3000) c=12;

else c= s / 250; /\* 按250分类 \*/

switch( c )

c

{ case 0: d=0 ; break ; /\* s<250 \*/ 0

case 1: d=2 ; break ; /\*250≤s<500\*/ ≥1

case 2: /\*500≤s<750\*/

case 3: d=5; break; /\*750≤s<1000\*/ <4

case 4: /\*1000≤s<1250\*/ ≥4

case 5: /\*1250≤s<1500\*/



```

case 6:                /*1500≤s<1750*/
case7: d=8 ; break ; /*1750≤s<2000*/      <8
case8:                /*2000≤s<2250*/      ≥8
case9:                /*2250≤s<2500*/
case10:               /*2500≤s<2750*/
case11: d=10 ; break; /*2750≤s<3000*/      <12
case12: d=15 ; break; /*3000≤s*/          ≥12
}
f=p*w*s*(1-d/100.0) ;
printf(“freight=%15.4f”, f) ;
}

```

运行情况： 输入： 10,20,300✓

结果： **freight= 58800.0000**

若用if 语句实现上述功能，程序为：

```
main( )  
{   int s;  
    float p, w, d , f ;  
    scanf("%f , %f , %d", &p, &w, &s);  
    if(s>=3000) d=15;  
    else if (s>=2000&&s<3000) d=10;  
    else if (s>=1000&&s<2000) d=8;  
    else if (s>=500&&s<1000) d=5;  
    else if (s>=250&&s<500) d=2;  
    else d=0;  
    f=p*w*s*(1-d/100.0) ;  
    printf("freight=%15.4f ", f) ;  
}
```

例5.6

求  $ax^2+bx+c=0$  方程的解

- 1、 $a=0$ ，方程不是二次方程
- 2、 $b^2-4ac=0$ , 有两个相等的实根
- 3、 $b^2-4ac>0$ , 有两个不相等的实根
- 4、 $b^2-4ac<0$ , 有两个共轭复根
- 5、如何判断实数为零（或大于零，或小于零）

特殊条件的处理

```
#include<math.h>
main()
{ float a,b,c,d,disc,x1,x2,
  realpart,imapart;
  scanf(“%f,%f,%f”,&a,&b,&c);
  printf(“The equation ”);
  if(fabs(a)<=1e-6)
    printf(“is not a quadratic”) ;
  else
  {
    disc=b*b-4*a*c;
    if(fabs(disc)<=1e-6)
      printf(“has two equal roots:
        %8.4f\n”,-b/(2*a));
```

```
    else if( disc>1e-6)
      { x1=(-b+sqrt(disc))/(2*a);
        x2=(-b-sqrt(disc))/(2*a);
        打印实根(略);
      }
    else
      { realpart=-b/(2*a);
        imagpart=sqrt(-disc)/(2*a);
        printf(“has complex roots:\n”) ;
        打印虚根(略);
      }
  }
}
```

有问题吗？

```
{
    if(fabs(a)<=1e-6)
        printf("is not a quadratic") ;
    else
        {disc=b*b-4*a*c;
        if(fabs(disc)<=1e-6)
            printf("has two equal roots:%8.4f\n",-b/(2*a));
        else if( disc>1e-6)
            { x1=(-b+sqrt(disc))/(2*a);
            x2=(-b-sqrt(disc))/(2*a);
            打印实根(略);
            }
        else
            { realpart=-b/(2*a);
            imagpart=sqrt(-disc)/(2*a);
            printf("has complex roots:\n") ;
            打印虚根(略);
            }
        }
}
```

习题5.7 给一个不多于5位的正整数,要求:①求出它是几位数;②按逆序打印出各位数字,例如原数为12345,应输出54321。

```
#include <stdio.h>
main()
{
    long int num;
    int n1,n2,n3,n4,n5,bits;
    scanf("%ld",&num);
    if (num>9999) bits=5;
    else if (num>999) bits=4;
    else if(num>99) bits=3;
    else if(num>9) bits=2;
    else bits=1;
    printf("bits=%d\n",bits);
```

```
n5=num/10000;
n4= (num-n5*10000)/1000;
n3=(num- n5*10000- n4*1000)/100;
n2=(num- n5*10000- n4*1000- n3*100)/10;
n1=num-n5*10000-n4*1000-n3*100- n2*10;
switch(bits)
{ case 5:printf("Reverse bits is: %d,%d,%d,%d,%d\n",
               n1,n2,n3,n4, n5); break;
  case 4:printf("Reverse bits is: %d,%d,%d,%d\n",
               n1,n2,n3,n4); break;
  case 3: printf("Reverse bits is: %d,%d,%d\n", n1,n2,n3);
           break;
  case 2: printf("Reverse bits is:%d,%d\n", n1,n2); break;
  case 1: printf("Reverse bits is: %d\n", n1);
}
}
```



运行情况: **Input 12345✓**

**Output bits=5**

**Reverse bit is: 5,4,3,2,1**

```
#include <stdio.h>
main()
{
    long int num;
    int n1,n2,n3,n4,n5,n[5],bits=5,flag=0;
    scanf("%ld",&num);

    n[0]=num/10000;
    n[1]=(num-=n[0]*10000)/1000;
    n[2]=(num-=n[1]*1000)/100;
    n[3]=(num-=n[2]*100)/10;
    n[4]=(num-=n[3]*10);

    for(i=0;i<5;i++)
    {if(n[i]==0&&flag==0) bits--;
    else flag=1;
    }

    //bits=(n5&&1)+(n4&&1)+(n3&&1)+(n2&&1)+(n1&&1);
    printf("bits=%d\n",bits);
    switch(bits)
    .....
}
```

## 要牢记

- ❖ 不要在关系运算符`==`,`!=`,`>=`,`<=`的两个符号之间插入空格;
- ❖ 注意“与”运算(`&&`)和“或”运算(`||`)都是由两个相同的符号组成;
- ❖ 注意不要将等号运算符用于实型数, 它们很少能精确地相等;
- ❖ 语句标号 (**case** 常量表达式) 应该是一个整型常量表达式;
- ❖ 不得对**switch**语句的两个语句标号使用相同的常量。

# 本章作业

5-1 有如下函数，写一程序，输入x，计算y值，并输出x和对应的y值（设x=-1.5, -0.5, 0.5, 1.5）。

$$Y = \begin{cases} -0.5 & (x \leq -1) \\ (x+1)^2 - 0.5 & (-1 < x \leq 0) \\ (x-1)^2 - 0.5 & (0 < x \leq 1) \\ -0.5 & (x > 1) \end{cases}$$

5-2 输入3个不相等的实数，要求按由大到小的顺序输出。

5-3 有4个窟窿，圆心分别为(3,2)，(-3,2)，(-3,-2)，(3,-2)，圆半径均为1m。4个窟窿的深度均为5m，4个窟窿以外无建筑物。在 $x \in [-4, 4]$ ， $y \in [-3, 3]$ 的范围输入若干点的坐标，求该点的高度（窟窿以外地面的高度为0，窟窿内的地面高度为-5m）以测试程序。打印出x,y,z的值。

提示： $(x-3)^2 + (y-2)^2$  可以写为 $\text{pow}(x-3,2) + \text{pow}(y-2,2)$ 。

$\text{pow}$ (幂底，幂次)为C的标准函数，需要在主函数前使用  
`#include<math.h>`

5-4 一家时装屋贴出了如下的季节性打折通知。写一个程序，计算并打印出打折前后顾客购买该商品花费的比较，分别给出几种小于500元的商品价格进行测试。

购衣金额（元）	折扣discount	
price	成衣(Mill cloth)	手工艺品(Handloom items)
0~100	0	5%
101~200	5%	7.5%
201~300	7.5%	10%
>300	10%	15%

可以使用key标识顾客所购的是成衣还是手工艺品：

```
int key;
printf("The cloth type which
    custmer is to purchase is:\n");
printf("1——Mill cloth;\n");
printf("2——Handloom items.\n");
scanf("%d",&key);
```

.....

```
if(key==1)
    if(price<=100) discount=0
    else if...
    ...
else
    if(price<=100) discount=0.05
    else if...
    ...
```

以下各题自己对答案，有问题咨询老师

## 5-5 判断对错

- a) 如果格式说明符的长度比实际数值的长度大，该数值将被右对齐打印。
- b) 格式说明`%5s` 只会给出要打印字符串的头五个字符。
- c) 表达式`!(x<=y)`与 `x>y`意义相同。
- d) 在进行表达式混合运算时，隐形转换是自动进行的。
- e) 用于从键盘接受一个字符的标准C函数是**`getchar`**。
- f) **`scanf`** 函数不能用于从键盘读入单个字符。
- g) **`printf`** 语句中的打印表列（变量表列）可以包含函数调用。
- h) 对于嵌套的if语句，最后的**`else`**与距离最近的且没有与其它**`else`**匹配的if配对。
- i) **`switch`** 语句总是可以用一系列**`if...else`**语句取代。
- j) 一个if可以与多个**`else`**匹配。
- k) **`swich`**语句中，任何**`case`** 分支只能有一条可执行语句。



## 5-6 填空

- a) 格式说明\_\_\_\_\_用于整形数的读和写。
- b) `printf ("%2d %c %4.2f",1234,'x',1.234);`的输出是\_\_\_\_\_。
- c) 假设 `int count=1275; float price=235.74; printf ("%2d\n%f", count, price);` 的输出是\_\_\_\_\_。
- d) `printf ("%8.4s","Cambridge");` 的输出是\_\_\_\_\_。
- e) 必须当两个操作数均为“真”时，关系运算符\_\_\_\_\_的运算结果为“真”。
- f) 当程序在**switch**语句中遇到\_\_\_\_\_时，会立刻从该结构中退出。
- g) 为使数据左对齐打印，必须在格式说明中使用符号\_\_\_\_\_。
- h) `!(x!=y)` 可以用\_\_\_\_\_代替。
- i) 使用 `(? :)` 的条件表达式在用于\_\_\_\_\_语句时，会使得编程简单。

5-7 找出下列程序段中的错误，如果有的话。

(a) `if(x+y=z && y>0)`

`printf(" ");`

(b) `if(code>1);`

`a=b+c`

`else`

`a=0`

(c) `if (p<0) || (q<0)`

`printf(" sign is negative");`

5-8 对于下列程序段，当假设(a)n=1;(b)n=0时，x和y的值是多少。

```
x=1;
```

```
y=1;
```

```
if(n>0)
```

```
    x=x+1;
```

```
    y=y-1;
```

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

**5-9** 不使用逻辑运算符，重写下列程序段。

(a) `if(grade<=59 && grade>=50)`

`second=second+1;`

(b) `if(number>100 || number<0)`

`printf(" Out of range");`

`else`

`sum=sum+number;`

**5-10** 简化下列复合逻辑表达式

(a) `!(x<=10)`

(b) `!(x==10||!((y==5)||(z<0)))`

(c) `!(x+y==z)&&!(z>5)`

(d) `!((x<=5)&&(y==10)&&(z<5))`

**5-11** 假设 $x=10$ ，说明下列逻辑表达式为“真”还是为“假”。

- a)  $x==10 \ \&\& \ x>10 \ \&\& \ !x$
- b)  $x==10 \ || \ x>10 \ \&\& \ !x$
- c)  $x==10 \ \&\& \ x>10 \ || \ !x$
- d)  $x==10 \ || \ x>10 \ || \ !x$

**5-12** 找出下列switch语句中的错误，如果有的话。  
假设：  $\text{int } x=1, y=2;$

- a) `switch(y);`
- b) `case 10;`
- c) `switch(x+y)`
- d) `switch(x) {case 2: y=x+y; break} ;`

# 答案

## 5-5 判断对错 5-6 填空

- a) True
- b) False
- c) True
- d) True
- e) True
- f) False
- g) True
- h) True
- i) True
- j) False
- k) False

- a) %d, 或%i
- b) 1234 x 1.23
- c) 1275  
235.740000
- d) Camb
- e) &&
- f) break
- g) -
- h) x==y
- i) if else

## 5-7 改错

(a) If(x+y==z && y>0)

printf(" ");

(b) If(code>1)

a=b+c;

else

a=0;

(c) If (p<0 || q<0)

printf(" sign is negative");

5-8

(a)  $n=1$

$x=2, y=0$

b)  $n=0$

$x=1, y=0$

5-9

(a)  $\text{if}(\text{grade} \geq 50)$

$\text{if}(\text{grade} \leq 59)$

$\text{second} = \text{second} + 1;$

(b)  $\text{if}(\text{number} < 0)$

$\text{printf}(\text{" Out of range"});$

$\text{else if}(\text{number} > 100)$

$\text{printf}(\text{" Out of range"});$

$\text{else}$

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



## 5-10

- (a)  $x > 10$
- (b)  $x < 10 \parallel x > 10 \parallel (y < 5 \ \&\& \ z \geq 0) \parallel (y > 5 \ \&\& \ z \geq 0)$
- (c)  $(x + y < z) \ \&\& \ (z \leq 5) \parallel (x + y > z) \ \&\& \ (z \leq 5)$
- (d)  $x > 5 \parallel y < 10 \parallel y > 10 \parallel z \geq 5$

## 5-11

- a) False
- b) True
- c) False
- d) True

## 5-12

- a) `switch(y) {}`
- b) `case 10:`
- c) Right
- d) `switch(x)`  
`{case 2: y=x+y; break;}`

# 第六章 循环控制

## (Cycle Control)

- 了解**goto**语句和**if** 语句构成的循环。
- 熟练掌握**for**循环语句、**while**循环语句和**do-while**循环语句，以及**break**语句、**continue**语句的使用。
- 熟练掌握循环结构程序设计的概念及其程序编制技术。

```
void main()
{
int i=1;
long int  sigma=1;
    while (i<=10)
    {
        sigma=sigma*i;
        i++;
    }
printf("10!=%ld\n", sigma);
}
```

$\text{sigma} = 1 * 2 * 3 * 4 * 5 * 6 * 7 * 8 * 9 * 10;$

sigma

$10! = 3628800$

思考：如果不使用long int，还可以使用什么变量类型定义sigma

循环结构的程序流程是：

在满足给定条件时，反复执行某个程序段。

C 语言中可以用以下语句来实现循环：

- (1) 用goto语句和 if 语句构成循环；
- (2) 用while语句；
- (3) 用do\_while语句；
- (4) 用for语句。

## 6.2 goto语句和goto语句构成的循环

### 一、 goto 语句（Goto Sentence）

goto 语句为无条件转向语句，其作用为：  
使程序的执行无条件地转移到指定处。

一般形式： goto 语句标号；

**执行过程：** 执行语句时，程序转移到以标号（定名规则与变量相同）为前缀的语句处继续执行。

如： 向前跳

```
goto label;
```

.....

.....

```
label: 语句;
```

向后跳

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100

如：     ✓ goto label\_1;     ✗ goto 123;

.....

.....

## 说明:

(1) C语言允许在任何语句前加一个语句标号，作为goto语句的目标。给语句加标号的形式：

语句标号：语句

(2) goto语句是一种非结构化语句，结构化程序设计方法**不提倡**使用goto语句，因为滥用goto语句将使程序流程无规律，可读性差。但也不是绝对禁止使用，只有在能够大大提高程序效率时才使用。

切记：不要从循环体外跳到循环体内！



## 二、用goto语句和 if 语句构成循环

例：计算10！并输出结果。

```
void main()
{
int i=1, sum=1;
loop: if (i<=10)
    {
        sum=sum*i;
        i++;
        goto loop;
    }
printf(“10!=%d\n”,sum);
}
```

运行结果： 10!=24320

```
void main()
{
int i=1; long sum=1;
loop: if (i<=10)
    {
        sum=sum*i;
        i++;
        goto loop;
    }
printf(“10!=%ld\n”,sum);
}
```

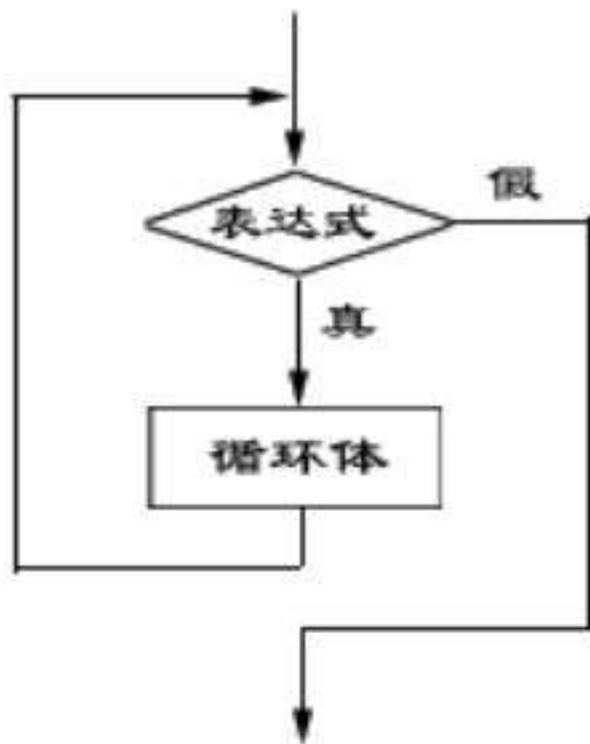
本应是： 10!=3628800

## 6.3 while循环语句

一般形式:

**while (表达式) 语句**

执行过程:



- (1) 先计算表达式的值；
- (2) 若表达式的值为真(非0)时执行循环体中的语句，然后重复上面步骤。若表达式的值为假(0)，则结束循环。

例：将上面计算10！并输出结果的程序用  
**while**语句写出：

//用goto语句

```
void main()
{
    int i=1;
    long int sum=1;
loop: if (i<=10)
    {
        sum=sum*i;
        i++;
        goto loop;
    }
    printf("10!=%ld\n",sum);
}
```

//用while循环语句

```
void main()
{
    int i=1;
    long int sum=1;
    while (i<=10)
    {
        sum=sum*i;
        i++;
    }
    printf("10!=%ld\n",sum);
}
```

## 例：计算 $y=x^n$

```
void main()
{
    int count=1,n;
    float x,y=1.0;
    printf("Enter the value of x and n : ");
    scanf("%f %d",&x,&n);
    while(count<=n)//测试表达式
    {
        y=y*x;
        count++;
    }
    printf("\nx = %f; n = %d; x to power n = %f\n",x,n,y);
}
```

Enter the value of x and n : 2.5 4

x = 2.500000; n = 4; x to power n = 39.062500

# 循环结构的术语

1. 循环条件: 是循环结构中的测试表达式。

如: `while(i<=10)`

2. 循环体: 是在每个循环周期均要执行一次的语句。循环体可以是任何语句, 简单语句、复合语句、空语句均可以。

如: 上例while语句中用花括号括起来的复合语句。

3. 循环控制变量: 在循环条件中控制测试条件为真或为假的变量。

如: 上例while语句中使用的变量 `i`。

## 注意：

(1)循环条件中的表达式一般是**逻辑表达式或关系表达式**，也可以是**算术表达式**（非0为真，0为假）。一般表达式中应含有循环控制变量。

**while (3)**

**while (0)**

虽然从程序设计的角度上说是不合理的，但却是合法的。

while (3) 语句	表示无限循环
while (0) 语句	表示不进入循环体

(2)要写出一个正确的循环结构，对控制变量要做三方面工作：

①控制变量赋初值；

②把控制变量写入正确的循环条件（测试表达式）；

③控制变量的更新、调整。

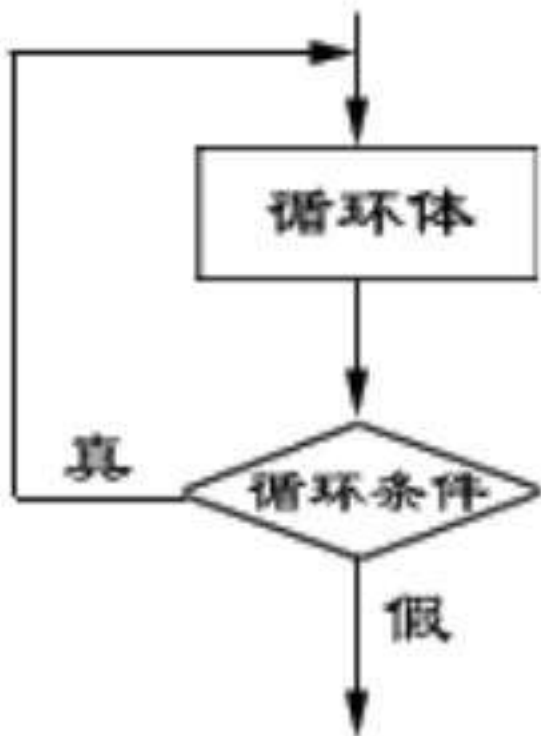
## 6.4 do-while循环语句

一般形式:

```
do  
    语句  
while(表达式);
```

分号不能少

执行过程:



- (1) 执行循环体中的语句;
- (2) 计算表达式, 测试循环条件, 为真(非0)则重复上面步骤, 为假(0)结束循环。



例：将上面计算10! 并输出结果的程序用do-while语句写出：

//用while循环语句

```
void main()
{
    int i=1;
    long int sum=1;
    while (i<=10)
    {
        sum=sum*i;
        i++;
    }
    printf( "10!=%ld\n" ,sum);
}
```

//用do-while循环语句

```
void main()
{
    int i=1;
    long int sum=1;
    do
    {
        sum=sum*i;
        i++;
    }while (i<=10);
    printf( "10!=%ld\n" ,sum);
}
```

# 将while和do-while循环做一下比较:

```
#include<stdio.h>
void main()
{
    int i; long sum=1;
    scanf( "%d" ,&i);
    while (i<=10)
    {
        sum=sum*i;
        i++;
    }
    printf( "%ld\n" ,sum);
}
```

```
#include<stdio.h>
void main()
{
    int i; long sum=1;
    scanf( "%d" ,&i);
    do
    {
        sum=sum*i;
        i++;
    }while (i<=10);
    printf( "%ld\n" ,sum);
}
```

输入: 9 ✓  
输出: 90

输入: 11 ✓  
输出: 1

输入: 9 ✓  
输出: 90

输入: 11 ✓  
输出: 11

比较上面两个程序，虽然结构是相同的，在输入值为有效值时结果相同，但当输入无效值时输出结果是不同的。为什么？

## while与do-while的主要区别：

while语句进入循环体之前，先测试循环条件，表达式必须为真；否则while循环不进入循环体，即循环体可能一次也不执行。

而do-while语句先执行循环体，然后测试循环条件。不管开始的时候测试循环条件是真是假，循环体都要执行一次。

## 6.5 for循环语句

一般形式：

不能省略

**for(表达式1; 表达式2; 表达式3) 语句(循环体)**

表达式1：变量初始化

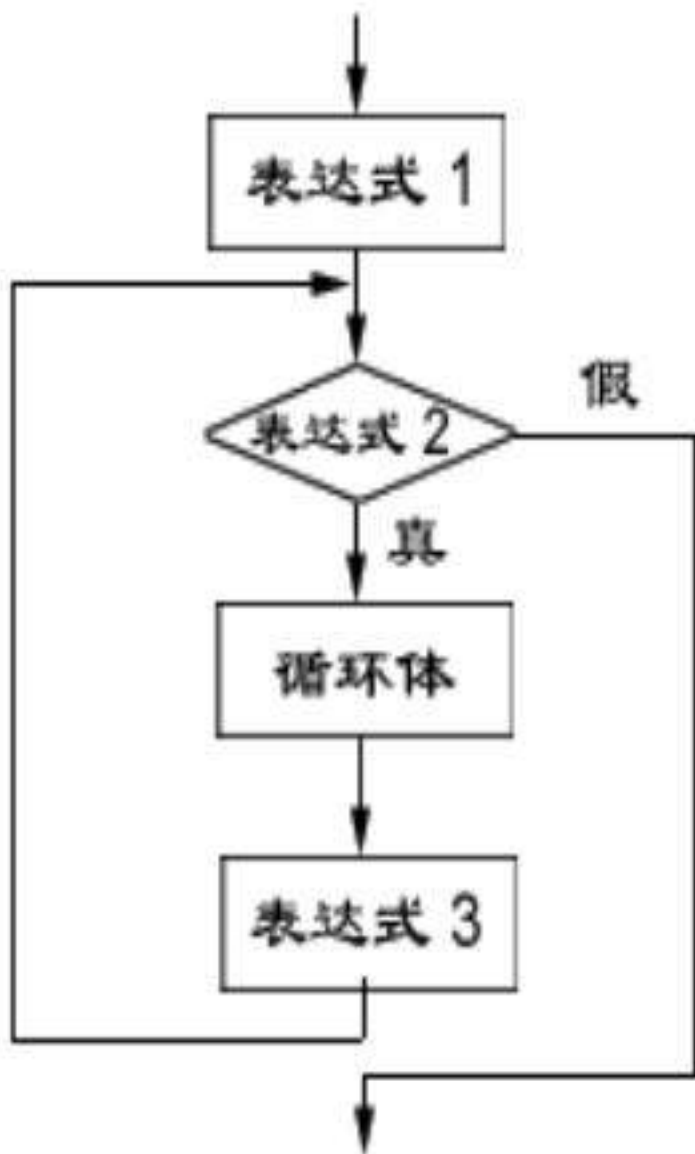
表达式2：循环控制条件测试

表达式3：控制变量更新

前面已经说明，要正确表达循环结构应注意三方面的问题：控制变量的初始化，循环条件的设置和控制变量的更新。

for语句在书写形式上正好体现了这种紧密的逻辑关系。

## 执行过程：



(1) 求出表达式1的值，若表达式1是逗号表达式，则依次计算；

(2) 对“表达式2”进行判断，为真转(3)，为假则退出循环；


(3) 执行循环体中语句；

(4) 执行“表达式3”，若表达式3是逗号表达式，则依次计算；

(5) 转向(2)。

例：将上面计算10! 的程序用 for语句写出：

```
main()
{
    int i;  long sum=1;
    for(i=1; i<=10; i++) sum=sum*i;
    printf( "10!=%ld\n",sum);
}
```

```
main()
{
    int i;  long sum;
    for(i=1,sum=1; i<=10; sum=sum*i, i++);
    printf( "10!=%ld\n",sum);
}
```

空语句

## 说明:

(1) **for** 循环的三个表达式起着不同的作用，  
根据需要可以省略。

表达式1: 用于进入循环体之前给某些变量赋初值。  
若省略，可在for语句前给变量赋初值。

```
main()
{
    int i;   long sum=1;
    i=1;
    for( ; i<=10; i++) sum=sum*i;
    printf( "10!=%ld\n" ,sum);
}
```

表达式2：决定循环的条件，若省略，则为无限循环。

如： for(i=0; ; i++) 语句	}	无限循环 (死循环)
for( ; ; ) 语句		
for(i=0; i<=10; i--) 语句		

表达式3：用于循环一次后对某些变量进行修改。  
若省略，可在循环体内对变量进行修改。

```
main()
{
    int i; long sum=1;
    for(i=1 ; i<=10 ; )
    { sum=sum*i; i++;}
    printf( "10!=%ld\n" ,sum);
}
```



(2) **for**语句功能很强，其中表达式1和表达式3可以是逗号表达式，但为增强程序的可读性，一般不要把与循环无关的东西放到**for**语句中。

**for**语句最简单、常用的形式如下：

for(循环变量赋初值； 循环条件； 循环变量增值) 语句

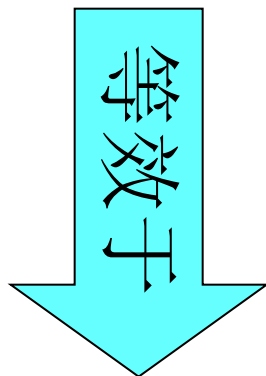
如： sum=1;

for(i=1 ; i<=10 ; i++) sum=sum\*i;

(3) **for** 语句最为灵活，它完全可以代替**while**语句。

如： `i=1;`

```
while(i<=10)
{
    sum=sum*i;
    i++;
}
```



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

## 6.6 三种循环语句的总结

(1)三种循环语句共同的特点是：当循环控制条件非零时，执行循环体语句，否则终止循环。

(2)循环体语句可以是任何语句，简单语句、复合语句、空语句均可以。

```
for(expression 1)
```

```
{
```

```
Setence 1;
```

```
Setence 2;
```

```
Setence 3;
```

```
}
```

```
if(e1)
```

```
{
```

```
S1;
```

```
S2;
```

```
}
```

```
else if(e2)
```

```
S3;
```

(3)while和for语句先判断循环控制条件，所以它们的循环体可能一次也不执行；而do-while语句后判断循环控制条件，它至少执行一次循环体。

(4)在循环条件中或循环体内必须有使循环趋于结束的语句，否则会出现死循环等异常情况。

(5)三种循环可以处理同一问题，一般情况可以互相代替。但在实际应用中，我们要根据具体情况来选用不同的循环语句。选用的一般原则如下：

•如果循环次数在执行循环体之前就已确定，一般用for语句；如果循环次数是根据循环体的执行情况确定的，一般用while语句或者do-while语句。

(循环次数确定例)

```
int i,n;  
long sum=1;  
scanf("%d",n);  
for(i=1;i<=n;i++)  
    sum=sum*i;
```

(循环次数不确定例)

```
char c  
do  
{ c=getchar();  
  printf("%c",c);  
}while(c!='q'&& c!='Q');
```

•当循环体至少执行一次时，用**do-while**语句；反之，如果循环体可能一次也不执行时，用**while**语句。

(循环体至少执行一次)

/\*只有当用户键入q或Q,才结束循环\*/

char c

do

```
{ c=getchar();  
  printf("%c",c);  
}while(c!='q'&& c!='Q');
```

(循环体可能一次不执行)

/\*只有不是q或Q的字符才被打印\*/

char c;

c=getchar();

while (c!='q'&& c!='Q');

```
{ printf("%c",c);
```

```
  c=getchar();
```

```
}
```

注意：  $!(a \&\& b) == !a \parallel !b$

$!(a \parallel b) == !a \&\& !b$

所以：  $c != 'q' \&\& c != 'Q'$

$\iff !(c == 'q' \parallel c == 'Q')$

## 6.7 循环的嵌套

例：在屏幕上打印一个8行7列的星号矩阵。

```
#include <stdio.h>
main()
{
    int i;
    for( i=0; i<7; i++ ) printf("*"); /*打印第1行星号*/
    printf("\n");
    for( i=0; i<7; i++ ) printf("*"); /*打印第2行星号*/
    printf("\n");
    .....
    for( i=0; i<7; i++ ) printf("*"); /*打印第8行星号*/
}
```

# 什么叫循环嵌套？

一个循环的循环体中套有另一个循环叫循环嵌套。这种嵌套过程可以一直重复下去。

一个循环外面包围一层循环称为二重循环。

一个循环外面包围二层循环称为三重循环。

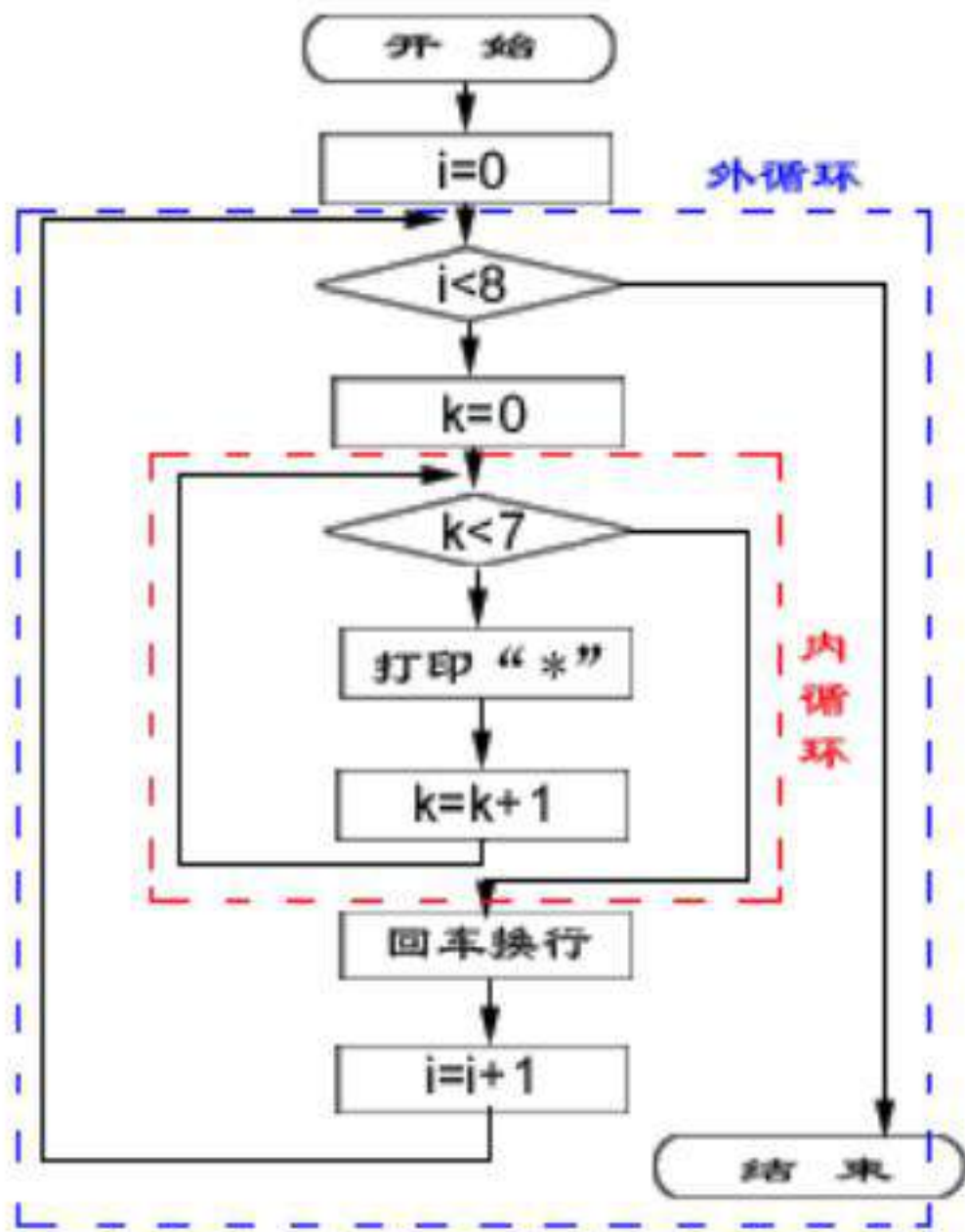
一个循环外面包围多层循环称为多重循环。

while、do-while、for可以互相嵌套，自由组合。



例：将上例（打印8行7列的星号矩形）用二重循环实现。

```
#include <stdio.h>
main()
{
    int i , k;
    for( i=0; i<8; i++ )
    {
        for( k=0; k<7; k++ )
            printf("*");
        printf("\n");
    }
}
```



例：在显示器上输出以下图案。

```
0      *
1     ***
2    *****
3   *********
4  ***********
5 *****
```

```
#include <stdio.h>
main()
{
    int i ,j, k;
    for( i=0; i<6; i++ )//i=1;i<7或者i=1;i<=6
    {
        for(j=0;j<5-i;j++) printf(" ");
        for( k=0; k<2*i+1; k++ ) printf("*");
        printf("\n");
    }
}
```

## 6.8 break 和continue语句

### 一、break 语句

一般形式：

```
break ;
```

功能：结束当前的循环语句。

解释：break 语句一般用在循环体的条件语句中，其作用是当某个条件满足时用break语句退出当前循环，不再继续执行剩余的几次循环。

例：打印半径为1~10的圆的面积，如果面积超过100，则不再打印。

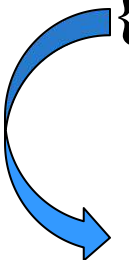
```
#include<stdio.h>
#define PI=3.14159//符号常量
main()
{
    int r;
    float area;
    for(r=1; r<=10; r++)
    {
        area=PI*r*r;
        if(area>100.0) break;//r=6
        printf("area=%f\n",area);
    }
    printf("now, r=%d\n",r);
}
```

错误！

**注意：**在嵌套循环中使用break语句，它只影响包含它的最内层循环，即程序仅跳出包围该break的那层循环。

例：编写程序，输出九九乘法表。

```
main()  
{  int i, j;  
    for( i=1; i<= 9; i++)  
    { for( j=1; j<= 9; j++)  
      { if( j>i ) break;  
        printf("%3d", i*j );  
      }  
      printf("\n");  
    }  
}
```



运行结果：

```
1  
2  4  
3  6  9  
4  8 12 16  
5 10 15 20 25  
6 12 18 24 30 36  
7 14 21 28 35 42 49  
8 16 24 32 40 48 56 64  
9 18 27 36 45 54 63 72 81
```

## 二、continue语句

一般形式：


```
continue ;
```

**功能：** 结束本次循环，进入下一个循环周期。

**解释：** 一旦执行了continue语句，程序就跳过循环体中位于该语句后的所有语句，提前结束本次循环并开始下一次循环。

例：计算用户输入的所有正整数的和，忽略负数，用户输入 **0** 时结束。

```
#include <stdio.h>
main()
{
    long data, sum = 0;
    do
    {
        printf("Please input : data=");
        scanf("%ld", &data);
        if( data < 0 ) continue;
        sum = sum + data;
    } while( data != 0 );
}
```



while(test-condition)

{ 注意continue语句用于各种循环语句的情况

if(-----)

continue;

-----

-----

}

c)

for(初始化; 测试表达式; 增量表达式)

{

-----

if(-----)

continue;

-----

-----

}

b)

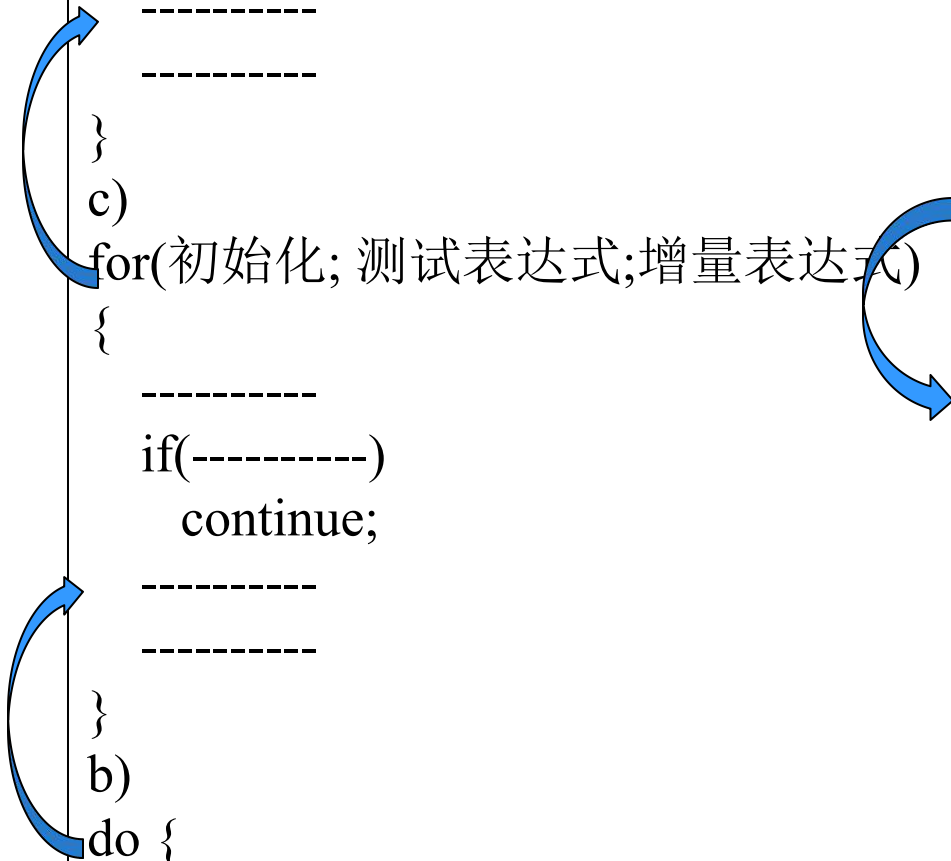
do {

-----

if(-----)

continue;

-----





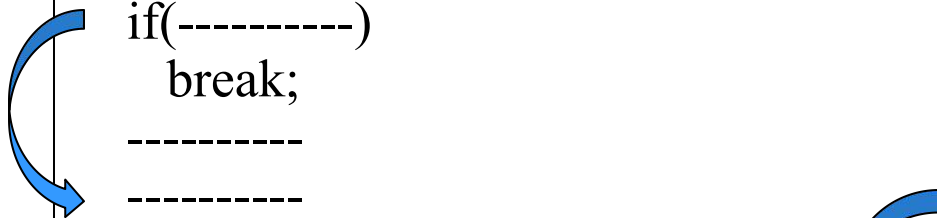
# 注意break语句用于各种循环语句的情况

c) for(初始化; 测试表达式; 增量表达式)

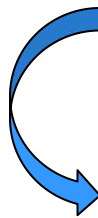
```
{  
    -----  
    if(-----)  
        break;  
    -----  
}
```



b) do {  
 -----  
 if(-----)  
 break;  
 -----  
} while(test-condition);  
-----



d) for(-----)  
{



## 6.9 程序举例

例6.6 用  $\pi/4 \approx 1 - 1/3 + 1/5 - 1/7 + 1/9 \dots$  求  $\pi$  的近似值。  
直到最后一项的绝对值小于  $10^{-6}$  为止。

```
#include "math.h"
main()
{ int sign=1;
  float n=1,t=1,sum=0, pi;
  while(fabs(t)>=1.e-6)//fabs()求绝对值的函数
  {   sum=sum+t;
      n=n+2;
      sign*= -1;
      t=sign/n;
  }
  pi=sum*4;
  printf("pi=%f\n",pi);
}
```

运行结果：  
pi=3.141594

斐波纳契Fibonacci数列：1，1，2，3，5，8...

第几个月	小兔子对数	中兔子对数	老兔子对数	总数
1	1	0	0	1
2	0	1	0	1
3	1	0	1	2
4	1	1	1	3
5	2	1	2	5
6	3	2	3	8
7	5	3	5	13

.....

**例6.7 求斐波纳契数列：1，1，2，3，5，8...  
的前20个数。**

**（该数列特点：第1、2两个数是1、1，从第3个数开始，每个数都是其前面两个数之和。）**

```
main()
{
    int f1=1, f2=1, i;
    for(i=1;i<=10;i++)
    { printf("%6d%6d\n",f1,f2);
      f1=f1+f2;
      f2=f1+f2;
    }
}
```

运行结果：

1	1
2	3
5	8
13	21
34	55
89	144
233	377
610	987
1597	2584
4181	6765

## 例6.7 求斐波纳契数列：1，1，2，3，5，8... 的前40个数。

```
#include<math.h>
main()
{
    long int f1=1, f2=1, i;
    for(i=1;i<=20;i++)
    { printf("%12ld  %12ld",f1,f2);
      if(fmod(i,2)==0) printf("\n");
      f1=f1+f2;
      f2=f1+f2;
    }
}
```

运行结果：

1	1	2	3
5	8	13	21
34	55	89	144
233	377	610	987
1597	2584	4181	6765
.....			

## 例6.8 判断m是否为素数。

方法：用 $2 \sim (m)^{1/2}$  之间的整数作为除数去整除m。

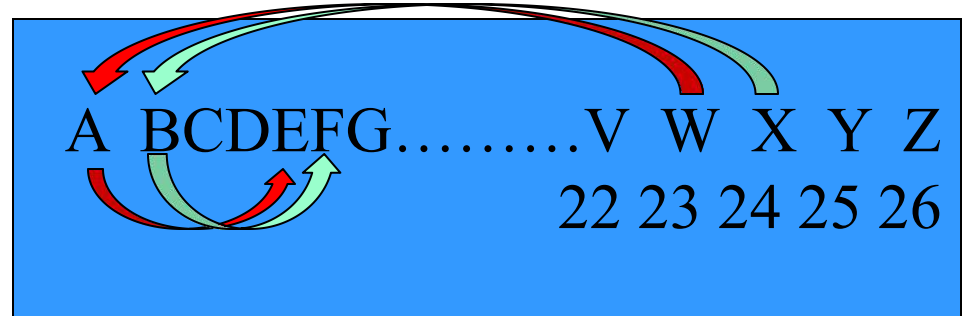
```
#include<math.h>
main()
{ int m, i ,k;
  scanf("%d",&m);
  k=sqrt(m); //m=83,1,2,3,4,5,6,7,8,9; 10,8
  for(i=2; i<=k ;i++)
    if(fmod(m,i)==0) break;
  if(i>k)
    printf("%d is a prime number.\n",m);
  else
    printf("%d is not a prime number.\n",m);
}
```

## 例6.9 求出100~200以内的所有素数。

```
main()
{
    int m , k,i ,n=0;
    for(m=101;m<=200;m=m+2)
    {
        k=sqrt(m);
        for(i=2;i<=k ;i++)
            if(m%i==0) break;
        if(i>=k+1)
            { printf(" %d",m); n=n+1;}
        if( n%10==0) printf("\n");
    }
    printf("\n");
}
```

**例6.10 译密码。**密码规律为：每个字母用其后  
第四个字母代替，26个字母循环排列。

```
#include "stdio.h"
main()
{
    char c,c1;
    while((c=getchar())!='\n')
    {
        if((c>='a'&&c<='v') || (c>='A'&& c<='V')) c=c+4;
        else if((c>='w'&&c<='z') || (c>='W'&&c<='Z')) c=c-22;
        printf("%c",c);
    }
}
```



China!  
Glmre!



## 习题6.3 (看书P120)

已知：一位数码 $a$  和要组成的最大重码数 $tn$ 的位数 $n$ 。

求：各重码数的累加和 $sn$ 。

算法：对给定的 $a$ 和 $n$ ，从 $i=1 \sim n$  依次作：

①求第 $i$ 个重码数 $tn = tn * 10 + a$ 。

（其中 $tn$ 的初值应是0）

②求累加和  $sn = sn + tn$ 。

```
main()
{
    int sn,a,n,i,tn;
    long sn,tn;
    printf("\na, n=");
    scanf("%d%d",&a,&n);
```

```
    for(tn=0,sn=0,i=1;i<=n;i++)
    {   tn=tn*10+a;
        sn+=tn;
    }
    printf("\na+aa+...=%ld",sn);
}
```

## 习题6.7 (看书P120)

已知：正整数  $1 \leq n \leq 1000$

求：其中的“完数”，即一个数的所有因子之和等于该数本身的数。

算法：对于  $n=2 \sim 1000$  的每一个数，进行：

① 求出该数的所有因子并累加于sum（不保存这些因子）。

（一个数n的所有因子的求法：只需用  $2 \sim n/2$  的所有数去除n,能整除的就是其因子。）

② 当能满足完数条件时（ $\text{sum}=n$ ），重新求出该数的所有因子并打印。

```
#include<math.h>
main()
{
    int sum,n,i;
    for(n=2;n<=1000;n++)
    {
        sum=0;
        for(i=1;i<=n/2;i++)
            if(fmod(n,i)==0) sum+=i;//求出n的因子,计算其和
        if(sum==n)
        {printf("\n%d : ",n);//打印完数
            for(i=1;i<=n/2;i++)
                if(fmod(n,i)==0)
                    printf("%d  ",i);//重新计算n的因子,并打印输出
            }
        }
    }
```

## 习题6.11 (看书P120~P121)

已知：正数 $a$ 和求 $a$ 的算术平方根 $x$ 的公式。

求： $a$ 的算术平方根 $x$ 。

算法：① 从键盘上输入 $a$ ，并假设 $a$ 的第一个根为 $x=a$ ；误差标志变量 $dx$ 。

② 重复做：

记下上一个根 $x_0=x$ ；

按公式求下一个根 $x$ ； $x=(x+a/x)/2$ ；

求两个根的误差绝对值 $dx=fabs(x-x_0)$ 。

③ 当两次求得的根之差的绝对值（精度） $dx \leq 0.00001$ 时停止。 $x$ 即为所求平方根。

```
#include "math.h"
main()
{
    float a, x, x0, dx;
    printf("\n a=");
    scanf("%f",&a);
    x=a;
    do
    {
        x0=x; //上次求出的根
        x=(x+a/x)/2; //本次求出的根
        dx=fabs(x- x0);
    } while(dx>0.00001);
    printf("\n x=%f",x);
}
```

# 要牢记

1. 勿忘在do...while语句后使用分号；
2. 在for语句的头表达式中使用逗号将产生错误；
3. 勿忘在while和do...while的循环体中使用增量语句；
4. 注意常见的用赋值运算符“=”代替关系运算符“==”的错误；
5. 不要对实数使用相等“==”的运算；
6. 对于在循环体中要进行反复运算的变量，要确认在进入循环体之前，该变量已被正确地赋初值；
7. 尽管允许对于变量初始化和增量运算使用算术表达式，但要清楚由此带来的舍入和截尾误差；
8. 在循环语句编程中使用缩进编排可以增强阅读性和易理解性。

# 本章作业:

编程，并上机调试以下各题：

**6-1 计算级数的和，精度要求为0.00001。三题任选一题。**

**a)采用下列级数方法计算欧拉数( $e=2.718281828\dots$ )的值：**

$$e=1+1/1!+1/2!+1/3!+\dots+1/n!$$

**b) 求和：**

$$SUM=1+(1/2)^2+(1/3)^2+(1/4)^2+\dots$$

**c) 输入任意角度x，计算：**

$$y=\cos(x)=1-x^2/2!+x^4/4!-x^6/6!+\dots$$

**注意：x使用弧度，例如： $\cos(60^\circ)=1-(3.1415926/3)^2/2!+\dots$**

## 6-2 在显示器上输出图案

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*

\*

(两个图案任选一个)



6-3 输入3个数，将其作为三角形的3边长，计算三角形的面积，输出3边长和三角形的面积。注意在程序里要对输入数据进行检查，对不能构成三角形的情况给出错误信息（构成三角形的条件为：任意两边之和大于第三边）。分别输入正确的和错误的（不能构成三角形的）数据进行检验。

## 判断对错

- a) **while**循环可以用来替代**for**循环，而不需在循环体中做任何变动。
- b) 在**for loop**语句中，控制变量的开始值必须小于其结束值。
- c) 在后判断的循环中（例如**do-while**循环），如果循环体执行了n次，则测试条件也计算了n次。
- d) **for**循环中的3个循环表达式必须使用逗号分开。
- e) **do...while**语句首先执行循环体，然后计算循环控制表达式。

## 答案

- a)F      b)F      c)T      d)F      e)T

# 第九章 预处理命令

1. 掌握编译预处理的概念和特点;
2. 掌握“文件包含”的概念和使用;
3. 掌握不带参数的宏定义及其使用, 了解带参数的宏定义及其使用。

# 概 述

## 一、编译预处理的概念

C语言允许在程序中使用几种特殊的命令（它们不是一般的C语句），在C编译系统对程序进行通常的编译之前，先对程序中这些特殊命令进行“预处理”，然后将预处理的结果和源程序一起再进行通常的编译处理，以得到目标代码。

## 二、主要预处理功能

(1)宏定义；    (2)文件包含；    (3)条件编译

## 9.1 宏定义

宏：是对正文进行代入或嵌入的一种功能。即从一字符流中取出某个字符串去代替源程序里的标识符。

### 一、不带参数的宏定义及宏替换

#### 1.一般形式：

```
#define 标识符 字符串
```

↑  
宏名

↑  
宏内容

无分号

作用：用标识符来代表一个字符串。

## 引例:

```
#define PI 3.1415926
```

---

```
main()
```

```
{
```

```
    float l,s,r,v;
```

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

```
    scanf("%f",&r);
```

```
    l=2.0*PI*r;//l=2.0* 3.1415926*r;
```

```
    s=PI*r*r;//s= 3.1415926*r*r;
```

```
    v=3.0/4*PI*r*r*r;// v=3.0/4* 3.1415926 *r*r*r
```

```
    printf("l=%10.4f\ns=%10.4f\nv=%10.4f\n",l,s,v);
```

```
}
```

其中 #define PI 3.1415926

作用是指定标识符PI来代表 “3.1415926”

## 说明:

- (1)宏名习惯用大写字母，以便与变量名相区别；
- (2)宏名用做代替一个字符串，不作语法检查；
- (3)宏定义的字符串不能以“;”结尾，字符串结束后一定要换行；
- (4)宏定义的有效范围为定义之处到`#undef`命令终止，如果没有`#undef`命令，则有效范围到本文结束；  
    `#undef`命令形式：

<code>#undef</code> 标识符
-------------------------
- (5)在进行宏定义时，可以引用已定义的宏名；
- (6)C语言允许宏定义出现在程序中函数外面的任何位置，但一般情况下它总写在文件的开头。



- 宏替换(宏展开)：用宏内容（字符串）原样代替程序中的所有宏名字的过程。

说明：(1)宏替换由编译程序预先进行；  
(2)宏替换范围是除字符串以外的所有宏名字；  
(3)若替换后文本串中仍含有宏名字，将再次进行替换，直到程序中不含宏名字为止。

#define PI 3.1415926	main()
#define R 3.0	{
#define L 2*PI*R	printf("L=%f\nS=%f\n",L,S);
#define S PI*R*R	}

第一次替换：printf("L=%f\nS=%f\n", 2\*PI\*R, PI\*R\*R);

二： printf("L=%f\nS=%f\n", 2\*3.1415926\*3.0, 3.1415926\*3.0\*3.0);

## 二、带参数的宏定义及宏替换

### 一般形式:

`#define` 宏名（宏形式参数表） 字符串

作用：宏替换时以实参数替代形参数。

```
#define PI 3.1415926
#define S(r) PI*r*r
main()
{
float r1=3.6, area;
area=S(r1); /* area=PI*r1*r1=3.1415926*r1*r1 */
printf("r=%f  area=%f\n",r1,area);
}
```

注意：宏替换后，程序的原意表达。

```
1 #define PF(x) x*x
2 #define PF(x) (x)*(x)
3 #define PF(x) ((x)*(x))
main()
{
    int a=2, b=3, c;
    c=PF(a+b)/PF(a+1);
    printf("\nc=%d ",c);
}
```

注意替换时不求值，  
只是字符串的原样替换

按第一种宏定义：  $c = a + b * a + b / a + 1 * a + 1$ ;

按第二种宏定义：  $c = (a + b) * (a + b) / (a + 1) * (a + 1)$ ;

按第三种宏定义：  $c = ((a + b) * (a + b)) / ((a + 1) * (a + 1))$ ;

### 三、带参数的宏替换与函数的主要区别

- (1)函数调用时，先求出实参表达式的值，然后代入形参。而使用带参数的宏只是进行简单的字符替换。
- (2)函数调用是在程序运行时处理的，分配临时的内存单元。而宏替换则是在编译时进行的，在展开时并不分配内存单元，不进行值的传递处理，也没有“返回值”的概念。
- (3)宏替换不占运行时间，只占编译时间，而函数调用则占运行时间。
- (4)函数中函数名及参数均有一定的数据类型，而宏不存在类型问题，宏名及其参数无类型。

## 程序举例：

```
#define MAX(x,y) x>y?x:y
main()
{
int n1,n2;
float f1,f2;
scanf("%d%d%f%f",&n1,&n2,&f1,&f2);
printf("maxi=%dmaxf=%f",MAX(n1,n2),MAX(f1,f2));
}
```

经预编译宏替换后的printf语句如下：

```
printf("maxi=%dmaxf=%f",n1>n2?n1:n2, f1>f2?f1:f2);
```

# 程序举例：

```
#include<math.h>
#define PR printf
#define NL "\n"
#define D "%d%d%d"
#define F "%f"
main()
{
    int a,b,c;
    float s,area;
    scanf(D,&a,&b,&c);// scanf("%d%d%d",&a,&b,&c);
    s=0.5*(a+b+c);
    area=sqrt(s*(s-a)*(s-b)*(s-c));
    PR(D NL,a,b,c);// printf("%d%d%d\n", a,b,c);
    PR(F NL,s); // printf("%f\n", s);
    PR(F NL,area); // printf("%f\n", area);
}
```

## 9.2 “文件包含”处理

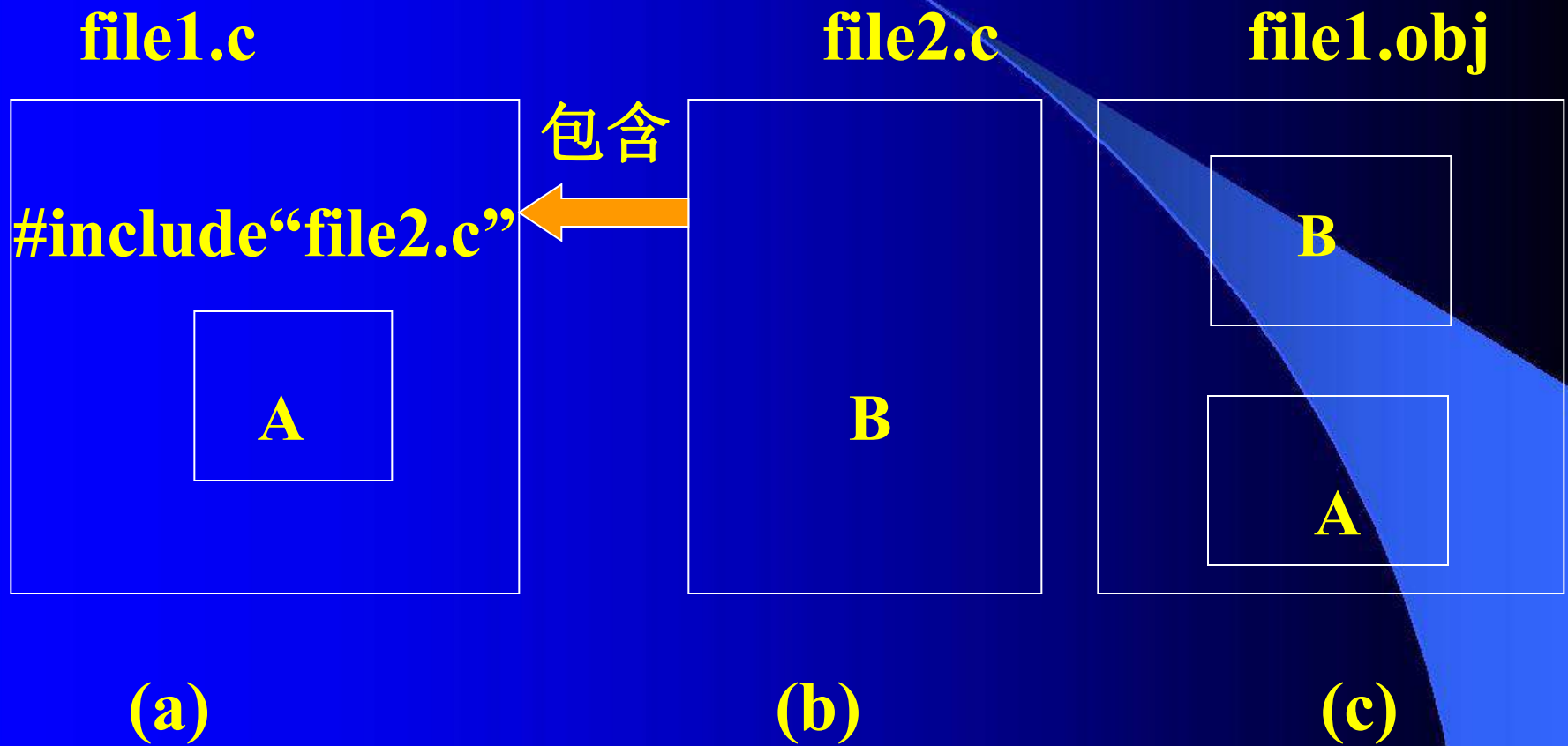
“文件包含”：是指一个源文件可以将另外一个源文件的全部内容包含进来。

C语言提供`#include`命令来实现“文件包含”的操作，其一般形式为：

`#include “文件名”` 或 `#include <文件名>`

**作用：**使编译系统把指定的被包含文件嵌入到带有`#include`的源文件中。

# “文件包含”示意图





## 包含文件的查找方法:

### **#include “文件名”**

先在当前工作目录中去查找，若找不到再到指定的标准目录中去查找。

如：对Turbo C编译系统，先在用户目录下查找，然后在TC\include文件夹中查找。

### **#include <文件名>**

直接到系统指定的标准目录中去查找。

如：对Turbo C编译系统，直接在TC\include文件夹中查找。例如：stdio.h, math.h

## 9.3 条件编译

条件编译：根据条件选择被编译的源程序行。

- 使用宏定义的标识符作为编译条件
- 使用常量表达式的值作为编译条件

### 一、使用宏定义的标识符作为编译条件

形式一：

```
# ifdef 标识符
    程序段1
# else
    程序段2
#endif
```

作用：当所指定的标识符已经被`#define`命令定义过，则在程序编译阶段只编译程序段1，否则编译程序段2。

例1:

```
#define DEBUG
```

```
#ifdef DEBUG
```

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

```
#endif
```

可用于进行程序的调试。

调试过程中，在程序前面加**#define DEBUG**  
调试完成后，将前面的**#define DEBUG**删除掉

## 例2:

```
#define ZD
```

```
.....
```

```
for(j=1;j<n-1;j++) //算j=1到j=n-2处的纵向棱边rcs
```

```
{ p5=p[i-1][j];p6=p[i][j];
```

```
  p7=p[i-1][j+1];p8=p[i-1][j-1];
```

```
#ifdef ZD //ZD was defined
```

```
  if(fabs(p5.z)<1.e-5&&(theta<theta_xia||theta>2.*PI-theta_sha))
```

```
    rcs+=rcszero;
```

```
  else
```

```
    rcs+=Bianyuanrcs(p5,p6,p7,p8,ii,ei,wavelength,kk,spin);
```

```
#else//ZD was not defined
```

```
  rcs+=Bianyuanrcs(p5,p6,p7,p8,ii,ei,wavelength,kk,spin);
```

```
#endif
```

## 形式二:

```
# ifdef 标识符  
    程序段1  
#endif
```

作用: 当所指定的标识符已经被**#define** 命令定义过, 则在程序编译阶段只编译程序段1。

## 形式三:

```
# ifndef 标识符  
    程序段1  
# else  
    程序段2  
#endif
```

作用: 当所指定的标识符**未被****#define** 命令定义过, 则在程序编译阶段只编译程序段1, 否则编译程序段2。

## 二、使用常量表达式的值作为编译条件

形式：

```
# if 表达式  
    程序段1  
# else  
    程序段2  
#endif
```

作用：当所指定的表达式为真（非零）时就编译程序段1，否则编译程序段2。

可以事先给定一定条件，使程序在不同的条件下执行不同的功能。

**例3：**对于一行字母字符，根据需要进行条件编译，使之能将字母全用大写输出，或者全用小写输出。

```
#include "stdio.h"
#define LETTER 1
main()
{
    char str[20]="C Language";
    int i=0;
    while((c=str[i])!='\0')
    {i++;
    #if LETTER
        if(c>='a'&&c<='z') c=c-32;
    #else
        if(c>='A'&&c<='Z') c=c+32;
    #endif
        printf("%c",c);
    }
}
```

# 本章作业:

9-1:输入两个整数,求它们相除的余数。用带参数的宏编程来实现。

9-2:试定义一个带参数的宏`swap(x,y,t)`,以实现两个整数`x`和`y`之间的交换,并利用它将一维数组`a[10]`和`b[10]`的值进行交换。( `a[10]`和`b[10]` 通过初始化方法赋值)



# 第七章 数 组(The Array)

- 掌握一维数组、二维数组的定义、初始化和引用。
- 掌握字符串和字符数组的概念及其处理函数。
- 掌握用数组进行编程的技术。

# 先看一个例子：

例：给出10个儿童的体重，要求计算平均体重并打印出低于平均体重的数值。

用变量来解决问题：

```
main()
{
    int w1, w2, w3, w4, w5, w6, w7, w8, w9, w10;
    int t;
    scanf( "%d%d%d%d%d%d%d%d%d%d", &w1,
           &w2, &w3, w4, &w5 , &w6, &w7, &w8,
           &w9, &w10);
```

```
t=(w1+w2+w3+w4+w5+w6+w7+w8+w9+w10)/10;  
if( w1 < t ) printf( "%d\n", w1 );  
if( w2 < t ) printf( "%d\n", w2 );  
if( w3 < t ) printf( "%d\n", w3 );  
if( w4 < t ) printf( "% d\n", w4 );  
if( w5 < t ) printf( "%d\n", w5 );  
if( w6 < t ) printf( "%d\n", w6 );  
if( w7 < t ) printf( "%d\n", w7 );  
if( w8 < t ) printf( "%d\n", w8 );  
if( w9 < t ) printf( "%d\n", w9 );  
if( w10 < t ) printf( "%d\n", w10 );  
}
```

## 用数组来解决问题:

```
main()
{
    int w[10]; /* 定义 1 个整型数组存放体重 */
    int t=0, i;
    for( i=0; i<10; i++ ) scanf( "%d", &w[i] );
    for( i=0; i<10; i++ ) t+=w[i];
    t = t/10;
    for( i=0; i<10; i++ )
        if( w[i] < t ) printf( "%d\n", w[i] );
}
```

# 数组及其相关概念

数组是一组有序的、类型相同的单元的集合，  
这些单元被称为数组的元素。



每个数组都有一个名字，称之为数组名。

为标识数组中的每个元素，需要对它们进行编号，称为数组元素的下标。

**注意：**C语言规定下标从0开始。

下标使数组元素在数组中的位置（或排列顺序）被唯一地确定下来；用数组名加上下标可以准确地访问数组中的某个元素。

如：w[0]代表数组w中的第一个元素

w[9]代表数组w中的第十个元素

说明：数组名代表数组的起始地址。

数组元素在内存中是连续存储的。

# 7.1 一维数组的定义和引用

## (One Dimension Array)

### 一、一维数组的定义

定义一维数组的一般方式：

类型说明符 数组名[常量表达式];

↑  
指明数组元素  
的数据类型

↑  
指明数组所含  
的元素个数

例如： `int a[10];` /\*定义的整型数组a含10个元素\*/

`float b[20];` /\*定义的实型数组b含20个元素\*/

# 说明:

- (1) 数组名的命名规则同变量名的命名规则一样;
- (2) 数组名后用方括号[], 不能用 ();
- (3) 常量表达式必须是大于0的整型常量表达式, 不能包含变量, 即其大小不依赖运行过程中变量值;
- (4) 定义数组时, 数组的长度必须是确定的, C语言不允许对数组的大小作动态定义。

例如:

```
#define size 10  
float a[size], b[size];  
int a[3*8+2]
```

~~int a(10);~~

~~int n=5;~~

~~int a[n]~~

~~char name[0];~~

~~float weight[10.3];~~

~~float array[-100];~~



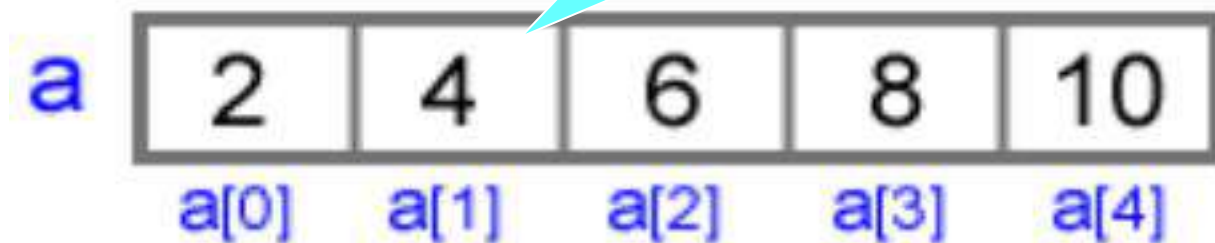
## 二、一维数组的初始化

### 1. 定义时初始化

方法：将初值依次写在花括号{ }内。

如： `int a[5]={ 2 , 4 , 6 , 8 , 10 };`

存储形式：



## 说明:

(1)如果定义一维数组时给出了全部元素的初值，则数组的下标允许省略。此时编译系统自动根据初始化数据的个数来确定数组的长度。

如： `int a[]={ 2 , 4 , 6 , 8 , 10 };`

`int a[];` ✗

(2)初始化的数据个数允许小于数组的长度，但不能大于数组的长度。当初初始化的数据个数（至少要有一个）少于数组的长度时，未初始化部分将被编译系统自动用 **0** 赋值。

如： `int a[5]={ 2 , 4 };`

相当于： `a[0]=2, a[1]=4, a[2]=0, a[3]=0, a[4]=0`

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

## 2 . 使用其它方法初始化

(1)利用赋值语句初始化

main()

```
{ char as[26],ch;  
    for(ch='A';ch<='Z';ch++) as[ch-'A']=ch;  
    .....  
}
```

(2)利用输入语句初始化

main()

```
{ char as[26]; int i ;  
    for(i=0; i<26; i++) scanf("%c",&as[i]);  
    .....  
}
```

### 三、一维数组的引用

数组元素类似于单个变量，可以自由地存取和参与各种运算。但是，它与一般变量相比，有以下特殊之处：

(1) 数组元素是通过数组名加上该元素在数组中的位置（即数组元素的下标）来访问的。

表示形式： 数组名[下标] 如： a[3]

其中， 下标可以是整型常量、整型变量或整型表达式。

如： int a[10] ;

float i=3 ;

a[i]=10 ;



(2) 数组元素的下标是从0开始的。

如果定义了a[n]，则可使用a[0]、a[1]、...、a[n-1]。但是，不能使用a[n]。

C语言编译系统不检查数组下标越界的错误，在编程的时候要注意避免这种错误。

如：int a[2];

~~scanf("%d,%d",&a[1],&a[2]);~~

(3) 数组元素的赋值是逐个元素进行的，不允许把一个数组作为一个整体赋给另一个数组。除了数组初始化外，也不允许用语句在花括号中列表的方式对数组整体赋值。

例1： `int a[5]={ 2 , 4 , 6 , 8 , 10 } , b[5] ;`

~~`b[5]=a[5];`~~

例2： `int a[5] ;`

~~`a[5]={ 2 , 4 , 6 , 8 , 10 } ;`~~

(4) 数组名`a`代表的是数组`a`在内存中的首地址，因此，可以用数组名`a`来代表数组元素`a[0]`的地址。

`scanf(“%d”,&a[0]);`  `scanf(“%d”,a);`

等价于

## 四、一维数组应用举例

例1：从键盘上输入10个实型数存入数组，然后按输入顺序的逆序输出这10个数。

```
main()
{ float a[10];
  int i;
  for(i=0 ; i<10 ; i++) scanf("%f",&a[i]);
  for(i=9 ; i>=0 ; i--) printf("%f",a[i]);
}
```

例7.2：用数组来处理求斐波纳契数列：

1, 1, 2, 3, 5, 8...的前20个数。

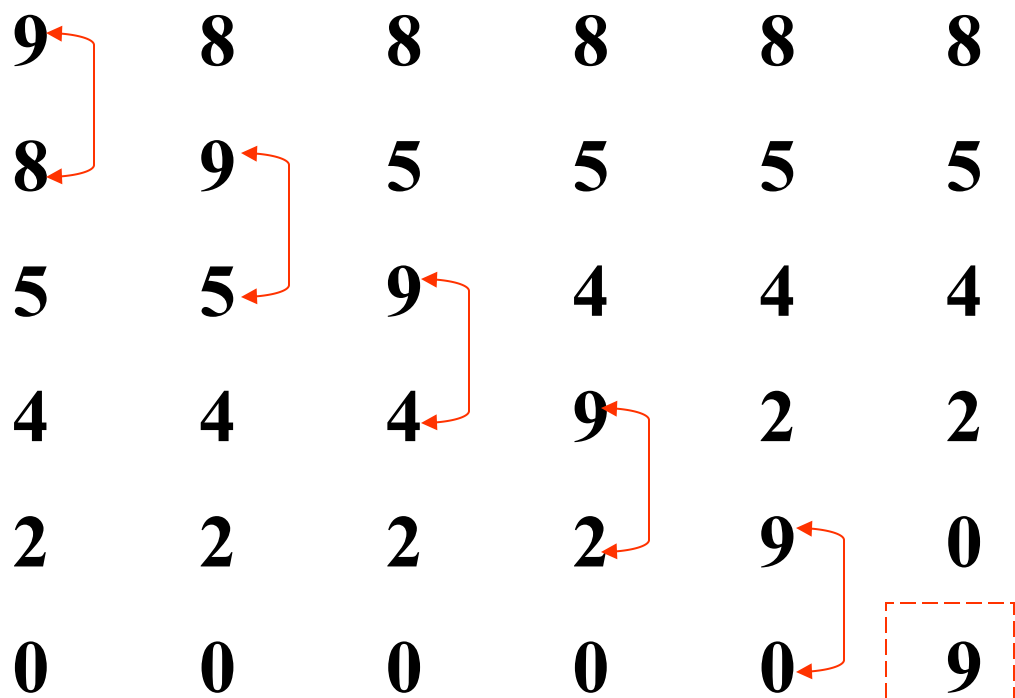
**数学表示：**  $f(0)=f(1)=1$ ,  $f(n)=f(n-2)+f(n-1)$

```
main()
{
    int f[20]={1,1};
    for(i=2 ; i<20 ; i++) /*f[2]是数组第3个元素*/
        f[i]=f[i-2]+f[i-1]; // f[2]=f[0]+f[1];
    for(i=0 ; i<20 ; i++)// f[3]=f[1]+f[2];
    {
        printf("%10d",f[i]);
        if(i%2==1) printf("\n"); /*每行打印2个数*/
    }
}
```



例7.3：用冒泡法对 $n=6$ 个数排序（由小到大）。

**冒泡法的思路是：**将相邻两个数比较，将小的调到前头。



轮数	剩数	比较次数
$j=1 \sim n-1$	$n-j+1$	$n-j$
1	6	5
2	5	4
3	4	3
4	3	2
5	2	1

第一轮排序，比较了5次，获得第一次结果。

```
main()
{
    int a[7], i , j , t ;
    printf("Input 6 numbers : \n");
    for(i=1 ; i<7 ; i++)
        scanf("%d",&a[i]);
    printf("\n");
    for(j=1 ; j<=5 ; j++) /* 控制5轮排序 */
        for(i=1 ; i<=6-j ; i++) /* 每轮排序比较次数*/
            if(a[i]>a[i+1])
                { t=a[i]; a[i]=a[i+1]; a[i+1]=t; }
    for(i=1 ; i<=6 ; i++)
        printf("%d",a[i]);
}
```

```
main()
{
    int a[6], i , j , t ;
    printf("Input 6 numbers : \n");
    for(i=0 ; i<6 ; i++)
        scanf("%d",&a[i]);
    printf("\n");
    for(j=0 ; j<=4 ; j++) /* 控制5轮排序 */
        for(i=0 ; i<5-j ; i++) /* 每轮排序比较次数*/
            if(a[i]>a[i+1])
                { t=a[i]; a[i]=a[i+1]; a[i+1]=t; }
    for(i=0 ; i<=5 ; i++)
        printf("%d",a[i]);
}
```

## 7.2 二维数组的定义和引用

先看一个例子：

某校近三年招收各专业本科生人数如下：

	计算机	电子	管理	数学
1999	90	40	80	30
2000	100	50	90	40
2001	95	45	100	50

要把这些数据组织起来，可以有两种选择：

- (1)按从左到右从上到下的顺序存入一个一维数组中。（查询困难）
- (2)每年用一个一维数组，把这些数据分别存入三个数组中。（增加一年数据困难）

# 一、二维数组的定义

- 定义二维数组的一般方式：

类型说明符 数组名[常量表达式1][常量表达式2];

如：     int a[3][4];  
          float b[5][6];

- 存储形式：

二维数组在内存中是按行的顺序存放的，即先存放第一行的元素，再存放第二行的元素。



说明： 二维数组除了维数比一维数组多一维外， 其它性质与一维数组是全部类似的。

看看下面写法是否正确？

① ~~int a[0][3];~~

用于定义数组长度的常量表达式的值必须是大于0的正整数。

② int i=3 , j=4 ;

~~int a[i][j];~~

定义数组元素的个数必须使用常量表达式，而不能使用变量。

## 二、二维数组的初始化

这里主要介绍定义时初始化，其它方法初始化和一维数组类似。定义时初始化有两种方法：

### (1)分行初始化

例如： **int a[2][3]={ {1 , 2 , 3 } , { 4 , 5 , 6 } };**

**int a[3][4]={ {1 , 2 , 3 , 4} , {5 , 6 } , {7} };**

<b>1</b>	<b>2</b>	<b>3</b>	<b>4</b>
<b>5</b>	<b>6</b>	<b>0</b>	<b>0</b>
<b>7</b>	<b>0</b>	<b>0</b>	<b>0</b>

### (2)省略掉内层的花括号

即按数组元素在内存中排列的顺序赋初值。

例如： **int a[2][3]={1 , 2 , 3 , 4 , 5 , 6 };**

## 说明:

(1) 对二维数组初始化时，可以只对部分数组元素初始化，未被初始化的数组元素将自动赋 **0**。

如： `int a[2][3]={5,6},{7,8};`      `int a[2][3]={5,6,7,8};`  
得到的数组为：  

5	6	0
7	8	0

  
得到的数组为：  

5	6	7
8	0	0

(2) 对二维数组初始化时，如果对全部元素都赋初值，或分行赋初值(每行至少一个数据)，则可以省略第一维数组长度。但是，第二维数组长度不允许省略。

如： `int a[][3]={1,2,3,4,5,6,7,8,9};`  
`int a[][4]={1,2},{3,4,5,6},{7}};`      } 系统按3行处理

~~`int a[][10]={1,0};`~~

~~`float f[2][]={1.1,2.2};`~~



### 三、二维数组的引用

与引用一维数组元素一样，是用下标法引用二维数组元素的。

表示形式：

数组名[行下标][列下标]
---------------

其中，行下标和列下标是整型常量、整型变量或整型表达式。其编号是从0开始的。

例如：若有 `int a[2*5][3*4], i=15;`

则使用 `a[3*3][0]`, `a[1][i-5]`都是合法的。

说明:

(1)数组名a代表的是a在内存中的首地址，因此可以用数组名a来代表数组元素a[0][0]的地址。

例如：若有 `int a[3][4]`, 则a[i][j]的地址为 $a+i*4+j$ ; a[0][0]的地址与a相同。

(2)二维乃至多维数组是若干个比其低一维的数组的集合。换句话说，二维乃至多维数组可以化成多个降低维数后的数组使用。

例如：若有 `int a[3][4]`，则等价于有了3个一维整型数组

a[0]: a[0][0], a[0][1], a[0][2], a[0][3]

a[1]: a[1][0], a[1][1], a[1][2], a[1][3]

a[2]: a[2][0], a[2][1], a[2][2], a[2][3]

## 四、二维数组应用举例

例1：使用数组保存“九九乘法表”，然后输出。

```
main()
{
    int a99[10][10], i, j ;
    for(i=1; i <10; i++)
        for(j=1; j<=i; j++) a99[i][j]=i*j;
    for(i=1; i<10; i++)
    {
        for(j=1;j<=i; j++) printf("%6d",a99[i][j]);
        printf("\n");
    }
}
```

## 例7.4： 矩阵的转置。（书中例7.4）

$$\mathbf{a} = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \quad \text{转换成} \quad \mathbf{b} = \begin{bmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{bmatrix}$$

```
main()
{
    int a[2][3]={ {1,2,3},{4,5,6} };
    int b[3][2], i,j;
    printf(" array a:  \n");
    for(i=0;i<=1;i++)
    {
        for(j=0;j<=2;j++)
        {
            printf("%5d",a[i][j]);
            b[j][i]=a[i][j];
        }
    }
    printf("\n");
}

printf("\n array b:  \n");
for(i=0;i<3;i++)
{
    for(j=0;j<2;j++)
        printf("%6d",b[i][j]);
    printf("\n");
}
```

## 例7.5：求3\*4数组中的最大值及其下标。

```
main()
{
    int max,row,colum,i,j;
    int a[][4]={ {1,2,3,4},{9,8,7,6},{-10,10,-5,2}};
    max=a[0][0];
    for(i=0;i<3;i++)
        for(j=0;j<4;j++)
            if(a[i][j]>max)
            {
                max=a[i][j];
                row=i;
                colum=j;
            }
    printf("max=%d,row=%d,colum=%d",max,row,colum);
}
```

## 7.3 字符数组和字符串

(Character Array and Strings)

### 一、字符数组(Character Array)

当数组的元素类型为字符型时，被称为字符数组。

如：char c[10]; 则c为字符数组。

字符数组的定义、初始化和元素的引用与一般的数组相同。

如：main()

```
{  
    char ch[5]={‘c’, ‘h’, ‘i’, ‘n’, ‘a’};  
    for(i=0; i<5; i++) printf(“%c”,ch[i]);  
}
```

## 说明:

(1)定义字符数组的类型说明符一般为char, 如:

`char c[10];` 由于字符型与整型的互相通用性,  
因此, 上面定义也可改为: `int c[10];`

(2)初始化赋值的字符个数允许少于字符数组的长度, 但不能多于字符数组的长度。当初始化赋值的字符数 (至少要有有一个) 少于字符数组的长度时, 未初始化部分将被编译系统自动用空字符 `'\0'` 赋值。

如: `int c[8]={ 'H' , 'e' , 'l' , 'l' , 'o' };`

则: `c[0]='H', c[1]='e', c[2]='l', c[3]='l', c[4]='o'`

`c[5]='\0', c[6]='\0', c[7]='\0'`

## 二、字符串与字符数组

字符串常量是用双引号括起来的一串字符，由系统自动加上一个字符串结束标志 ‘\0’。它占内存，但不记入字符串长度。

如：“china” 占内存6个字节，但字符串长度是5

在C语言中用字符数组来处理字符串，每个数组元素存放一个字符型数据。

字符型数组可用一般数组的初始化方式初始化外，还可以使用字符串常量初始化：

如：char message[]={“Hello”}; ✓  
或 char message[]=“Hello”; ✓

（这时，字符数组长度是6，字符串长度是5）



**注意：** 用一般初始化方式：

```
char message[]={‘H’, ‘e’, ‘l’, ‘l’, ‘o’};
```

结果： 

‘H’	‘e’	‘l’	‘l’	‘o’
-----	-----	-----	-----	-----

用字符串常量初始化：

```
char message[]={"Hello"};
```

结果： 

‘H’	‘e’	‘l’	‘l’	‘o’	‘\0’
-----	-----	-----	-----	-----	------

**说明：** 字符数组本身并不要求它的最后一个元素一定是 ‘\0’， 例如： `char ch[2]={‘A’, ‘B’};` 是合法的。

为了与用字符串常量对字符数组赋初值的处理方法一致，在字符数组中也常常人为地加上一个值为 ‘\0’ 的元素。

如： `char message[]={‘H’, ‘e’, ‘l’, ‘l’, ‘o’, ‘\0’};`

## 例7.7 输出一个钻石图形

main()

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

```
int i,j;
```

```
for(i=0;i<5;i++)
```

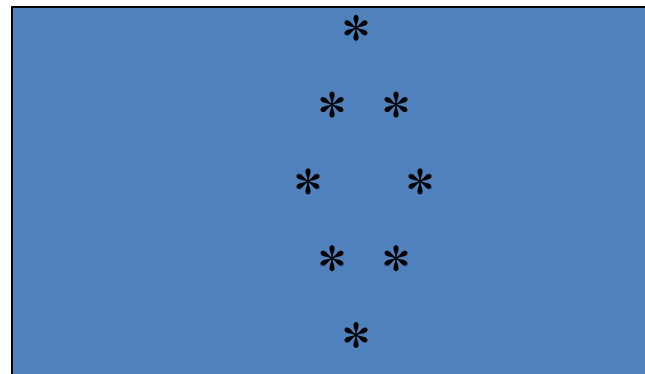
```
{ for(j=0;j<5;j++)
```

```
    printf("%c",diamond[i][j]);
```

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

```
}
```

```
}
```



### 三、字符数组的输入输出

字符数组的输入输出可以有以下两种方法：

#### (1) 逐个字符输入输出

用格式符 “%c” 输入或输出一个字符。

#### (2) 将整个字符串一次输入或输出

用格式符 “%s” 输入输出字符串。

例如：`char c[] = "Hello" ;`  
`printf( "%s" , c) ;`

**注意：** (1) 输出字符不包括结束符 ‘\0’。

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

(3)用 “%s”格式符输出字符数组时，遇到第一个 ‘\0’，输出就结束。

(4)用 “%s”格式符输入字符串时，scanf函数的输入项是字符数组名时，不要再加地址符&，因为C语言中数组名代表该数组的起始地址。

如： char str[10]; ~~scanf(“%s”,&str);~~  
scanf(“%s”,str);

(5)用 “%s”格式符输入字符串时，scanf函数的输入项是字符数组名，从键盘输入的字符串应短于已定义的字符数组的长度。系统自动在后面加个 ‘\0’结束符。

如： char str[6]; scanf(“%s”,str);

<b>‘a’</b>	<b>‘b’</b>	<b>‘c’</b>	<b>‘d’</b>	<b>‘e’</b>	<b>‘\0’</b>
------------	------------	------------	------------	------------	-------------

```
main()
{ char a[6]; int i=0;
  while(i<=5)
  { scanf("%c",&a[i]);
    i++;}
}
```

a	a[0]
b	a[1]
c	a[2]
	a[3]
c	a[4]
d	a[5]

输入数据:  
abc cdg

```
main()
{ char a[6]; int i=0
  scanf("%s",a);
  printf("%s",a);
}
```

a	a[0]
b	a[1]
c	a[2]
\0	a[3]
\0	a[4]
\0	a[5]

输入数据:  
abc cdg

(6)利用scanf函数输入多个字符串时，是以空格、回车、跳格(tab)分隔。因此不要试图利用scanf函数从键盘输入How are you? 赋给一个字符串。

例如，如果要输入How are you? 给一个字符串，则必须：

```
char str1[5],str2[5],str3[5];
```

```
scanf("%s%s%s",str1,str2,str3);
```

<b>'H'</b>	<b>'o'</b>	<b>'w'</b>	<b>'\0'</b>	
------------	------------	------------	-------------	--

<b>'a'</b>	<b>'r'</b>	<b>'e'</b>	<b>'\0'</b>	
------------	------------	------------	-------------	--

<b>'y'</b>	<b>'o'</b>	<b>'u'</b>	<b>'?'</b>	<b>'\0'</b>
------------	------------	------------	------------	-------------

## 四、字符串函数

### (一)、字符串输入输出函数

#### 1.字符串输入函数gets()

**使用形式：**

<code>gets(字符数组)</code>
-------------------------

**功能：** 从终端输入一个字符串(直到回车键)  
到字符数组中。

例如： `char str[20];`

`gets(str);`

若从键盘输入： Hello World!

则将字符串 “Hello World!”送到字符数组str中。

## 2.字符串输出puts()函数

使用形式:

**puts**(字符数组)

**功能:** 将一个字符串输出到终端。在输出时, 将字符串结束标志 ‘\0’ 转换成 ‘\n’, 即输出完字符串后自动换行。

例如: `char str[]={“China\nBeijing”};`

`puts(str);`

输出: **China**

**Beijing**

**注意:** 使用puts和gets函数前, 要用预编译命令:  
**#include “stdio.h”。**



**注意：** 使用这些函数前，要用#include “string.h”

## (二)、字符串处理函数

### 1. 求字符串长度strlen()函数

**使用形式：** `strlen(字符数组)`

**功能：** 计算字符串长度，函数值是字符串中 ‘\0’ 之前的字符个数（不包括 ‘\0’ 字符）。

例如： `char str4[20]={ “ab\n\\012/^\\” } ;`

`printf(“%d”,strlen(str)) ;`

输出： **10**      **(a,b,\n,\\,0,1,2,/,\\,”)**

`char str5[20]={ “ab\n\\0y\\012/^\\” } ;`

`printf(“%d”,strlen(str)) ;`

输出： **3**

## 2. 字符串比较函数strcmp()

**使用形式:** `strcmp(字符串1, 字符串2)`

其中，字符串可以是字符串常量或字符数组。

**功能:** 比较两个字符串。

**比较的原则:** 对两个字符串，从左至右逐个将对应字符按其 ASCII 码值大小相比较，直到出现不同的字符或遇到 ‘\0’ 为止。

若设  $n = \text{strcmp}(\text{字符串1}, \text{字符串2})$ ;

则: 
$$n = \begin{cases} n > 0; & \text{串1} > \text{串2} \\ n = 0; & \text{串1} = \text{串2} \\ n < 0; & \text{串1} < \text{串2} \end{cases}$$

例如：char str1[20], str2[10] ;

（将str1[20]和str2[10]赋初值）

if(strcmp(str1,str2)>0) puts(str1);

**注意： 不能写成 if(str1>str2)**

n=strcmp("China","Korea");

'C'=067<'K'=075

"China"<"Korea", n<0

同理：

"China">"Canada", n>0

"computer">"compare", n>0

"temperature"="temp"

### 3.字符串连接strcat()函数

**使用形式:**

<b>strcat</b> (字符数组1, 字符串2)
-----------------------------

**功能:** 连接两个字符串, 把字符串2连接到字符串1的后面, 连接后的字符串放在字符数组1中。

**说明:**

- (1)字符数组1必须足够大以便能够容纳字符串2。
- (2)连接时只在新串最后保留一个 ‘\0’。
- (3)字符串2可以是字符串常量或字符数组。

## 4. 字符串复制strcpy()函数

**使用形式:** `strcpy(字符数组1, 字符串2)`

**功能:** 将字符串2复制到字符数组1中  
(其后自动加一个 ‘\0’).

**说明:** (1)字符数组1必须足够大以便能够容纳被复制的字符串。

(2)不能用赋值语句将一个字符串常量或字符数组直接赋值给一个字符数组。

例如: `char str1[20], str2[20];`  
`strcpy(str1, "hello world");`  
`strcpy(str2, str1);`

例如: `char str1[20], str2[20];`  
~~`str1 = {"hello"};`~~  
~~`str2 = str1;`~~

# Strcpy使用举例:

```
#include <iostream.h>
#include<fstream.h>
main()
{char filewing[20]="wing.txt",filefuse[20]="fuselage.txt",filetail[20]="tail.txt";
    cout<<"请输入飞机部件类型 "<<"\n";
    cout<<"1----机翼"<<"\n";
    cout<<"2----机身"<<"\n";
    cout<<"3----尾翼"<<"\n";
    cin>>numpart;
    switch(numpart)
    case 1: {strcpy(filedes,filewing); break;}
    case 2: {strcpy(filedes,filefuse); break;}
    case 3: {strcpy(filedes,filetail); break; }
    case 4: default;
    ifstream in(filedes,ios::in);
    if(!in){cout<<"cannot open Inputfile "<<"\n";exit(1);}
}
```

## 5.将字符串中大写字母换成小写字母**strlwr()**函数

使用形式:

**strlwr**(字符串 )

## 6.将字符串中小写字母换成大写字母**strupr()**函数

使用形式:

**strupr**(字符串 )

## 五、字符数组应用举例

例1：(例题7.8)统计字符串中的单词数。

```
#include <stdio.h>
main()
{
    char string[81],c;
    int i,num=0,word=0;
    gets(string);
    for(i=0;string[i]!='\0';i++)
        if(string[i]==' ') word=0;
        else if(word==0)
            {word=1;num++;}
    printf("\nThere are %d words.", num);
}
```



I am a boy



word=0

N

$s[i] \neq '\backslash 0'$

Y

$s[i] == ' '$

Y

word=0  
(后面出现新单词)

N

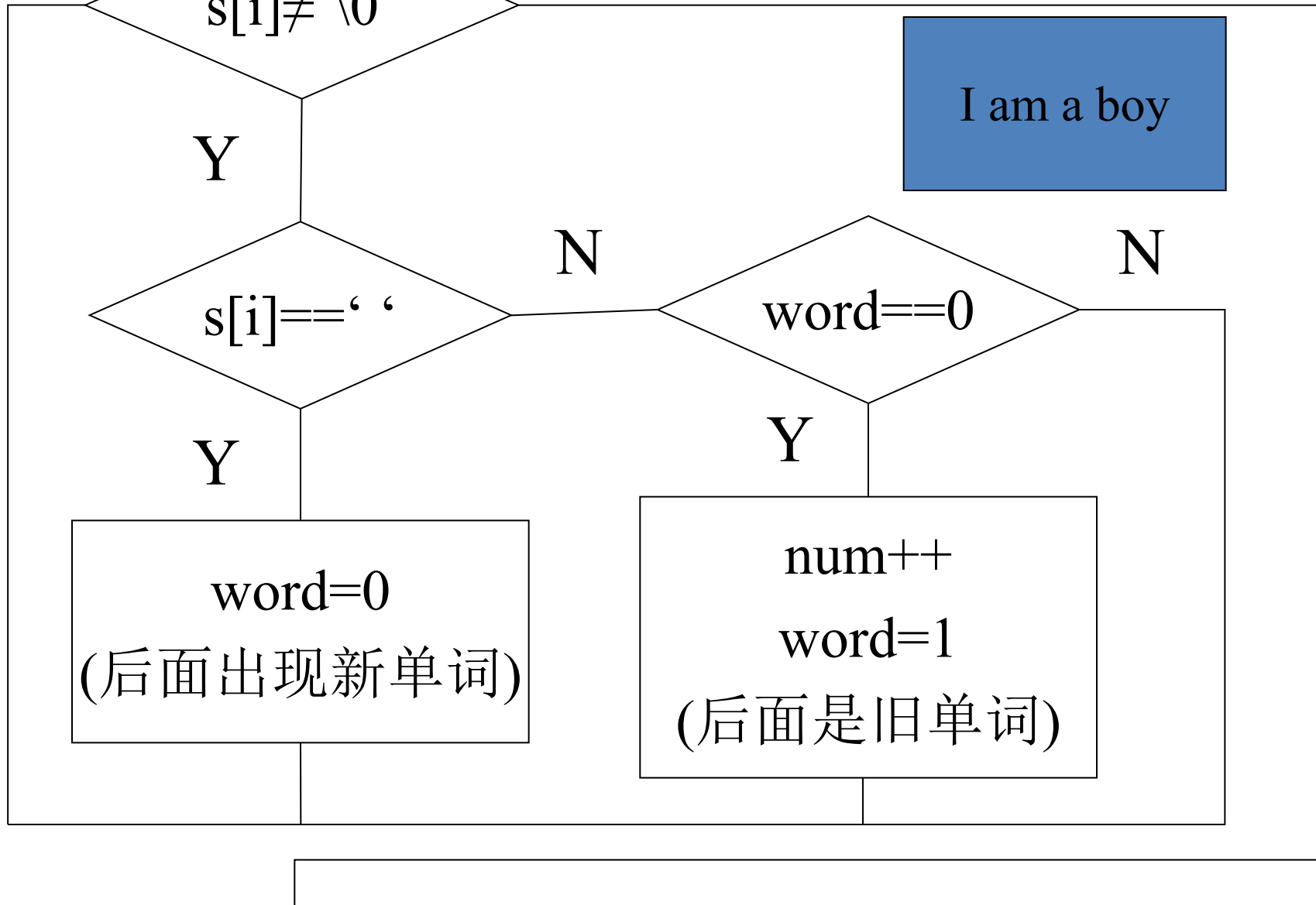
word==0

Y

num++  
word=1  
(后面是旧单词)

N

I am a boy



## 学习数组这一章要注意的几个问题：

1. 在C语言中数组的下标是从0开始；
2. C语言不进行下标的越界检查，不论是在编译阶段还是在运行阶段，这个问题交给程序员。这样就给初学者带来一定的风险。
3. 数组名是地址，这在字符串处理中已经看到，这个我们要先强记，以后在讲指针时再详细讲。
4. 字符数组在定义时必须考虑到串结束符的位置，因为它要占一个字符的位置。
5. 要注意数组初始化的方法与简单变量的区别，特别是字符数组的初始化方法。
6. 字符串的操作有许多专用函数，请注意掌握并运用。

## 本章作业： 编写并上机调试下列习题：

7-1 从计算机屏幕上任意输入10个实数，求出其中的最大值、最小值和10个数的算术平均值，并按下列格式在屏幕上打印输出（不得照抄冒泡法）。

**The maximum=**

**The minimum=**

**The average=**

7-2 按以下提示对矩阵a赋初值（不得使用初始化方法赋初值），并将转置后的矩阵b打印输出。

$$a = \begin{pmatrix} 1 & 0 & 0 & 0 \\ -1 & 1 & 0 & 0 \\ -1 & -1 & 1 & 0 \\ -1 & -1 & -1 & 1 \end{pmatrix}$$

$$b = \begin{pmatrix} 1 & -1 & -1 & -1 \\ 0 & 1 & -1 & -1 \\ 0 & 0 & 1 & -1 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

- **7-3 以下两题任选一题**
- 1) 求 $1!+2!+3!+\dots+10!$ 的和
- 2) 一个整数，它加上100后是一个完全平方数，该数加上268又是另一个完全平方数，请问该数是多少？（提示：1.在10万以内判断;2.如果一个数的平方根的平方等于该数，这说明此数是完全平方数）

**7-4** 在字符数组中存一段英文文章，按书上例6.10译成密码，取你的学号的尾数为字符的ASCII码的移动值（尾数为0时，取10），保持字母的大小写，除英文字符外，其它符号不译码（按下面提示）。并将原文和译码后的文字打印输出。

//字符数组赋初值

```
char s1[][80]={{"That may change in the near future."},
{"With it, two people are talking can see each other."}},s2[2][80];
for(i=0;i<2;i++)
for(j=0;s1[i][j]!='\0';j++)
{if(s1[i][j]>='a'&& s1[i][j]<='z')
    {s2[i][j]=s1[i][j]+n;
    if(s2[i][j]>'z') s2[i][j]-=26;}
else if(.....) {.....}
else s2[i][j]=s1[i][j];
}
```

# 第八章 函 数(The Function)

- 熟练掌握函数的定义、调用、返回值的方法及参数传递方式
- 掌握函数使用的常用方法
- 了解变量的作用域和变量的存储类型

# 8.1 概 述

## 一、C函数的概念

将一个C程序分为若干模块，每个模块实现一个特定的功能，在C语言中用函数来实现模块的功能。

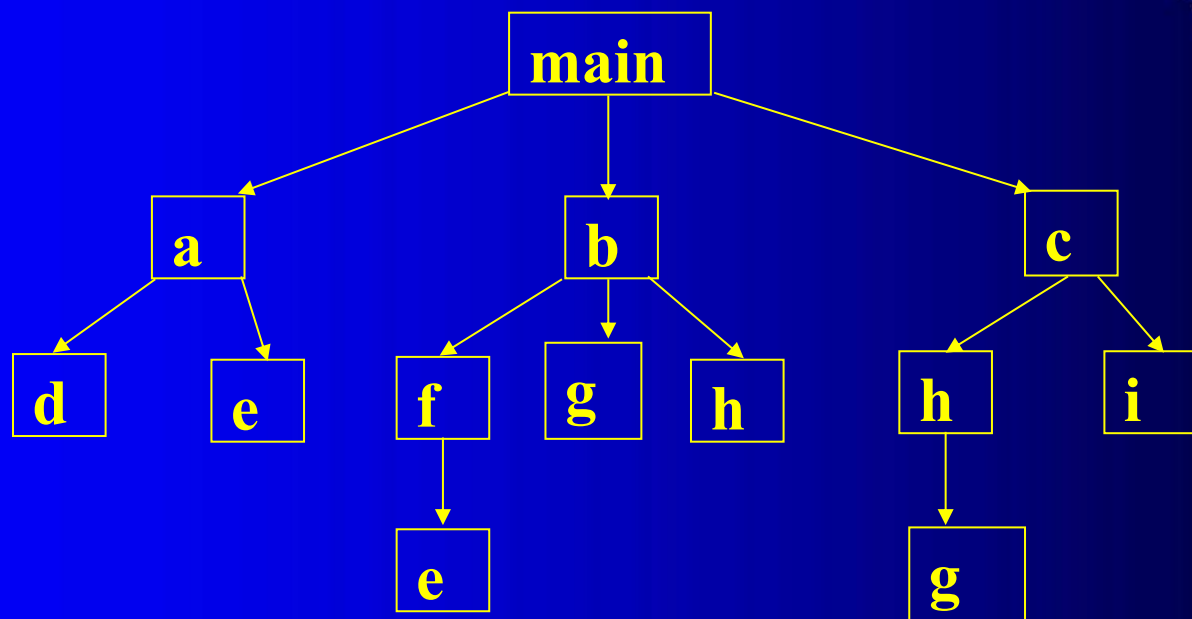
函数：具有某种功能的独立程序段。

(1)从程序设计方法看C函数：它是实现模块化程序设计的语法元素。

(2)从C语言中程序的组成方式看C函数：它是C程序的基本组成单位。

## 二、C函数与C程序结构

- (1) C程序由一个主函数（**main**函数）和若干个子函数构成；
- (2) 主函数调用子函数；
- (3) 子函数在定义时是并列的；
- (4) 子函数可相互调用，也可被多次调用。





```
#include<stdio.h>
#include<math.h>
void main()
{
char prime(int);/*函数声明或定义函数原形(Function
Prototype)*/
int n;
printf("\nInput an integer:");
scanf("%d",&n);
if(prime(n)=='g')//函数调用
    printf("\n%d is a prime.",n);
else
    printf("\n%d is NOT a prime.",n);
}
```

## //函数定义(Function Definition)

**char prime(int n)**//函数类型、名称，形式参数的定义

**{**//函数体定义

**char flag='g';**

**int i,k;**

**k=sqrt(n);**

**for(i=2;i<=k;i++)**

**if(fmod(n,i)==0)**

**{**

**flag='r';**

**break;**

**}**

**return flag;**//返回值

**}**

## 函数调用例:

```
void main()                /* 主函数 */
{
    print_star();          /*调用print_star函数打印****/
    print_message();       /*调用print_message函数写字*/
    print_star();          /*调用print_star函数打印****/
}

void print_star()          /*定义print_star函数*/
{
    printf("\n*****");
}

void print_message()       /*定义print_message函数*/
{
    printf("\n Hello! ");
}
```

运行结果: \*\*\*\*\*

**Hello!**

\*\*\*\*\*

### 三、C函数的特点

- (1)一个源文件由一个或多个函数组成，它是一个独立编译单元。
- (2)一个C程序由一个或多个源文件组成。

源文件1：主函数

```
#include "JPcal3.h"  
#include "PBcal.h "  
main(){...}
```

源文件2：函数JPcal3.h

```
#include "Complex.h"  
class CEq{...}
```

源文件3：函数Complex.h

```
#include "Vector.h"  
class CComplex{...}
```

源文件4：函数Vector.h

```
#include <stdio.h>  
#include <math.h>  
class CVector{...}
```

(3)C程序执行总是从main函数开始，调用其他函数后流程回到main函数，在main函数中结束整个程序的运行。

(4)函数不能嵌套定义，但可以互相调用。  
注意不能调用main函数。

## 四、函数的分类

### 1. 从函数的参数形式看，函数可分为两类：

①无参函数；      ②有参函数

有参函数例：(输出两数中大者)

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

```
int max( int x, int y ); //定义函数原形
```

```
void main( )
```

```
{
```

```
    int num1,num2,a;
```

```
    scanf("%d,%d",&num1,&num2);
```

```
    a = max(num1,num2);
```

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

```
}
```

```
int max( int x, int y )
```

```
{
```

```
    int z;
```

```
    if(x>y) z=x;
```

```
    else z=y;
```

```
    return z;
```

```
}
```

## 2. 从用户使用的角度看，函数有两种：

- ①标准函数(库函数)； ②用户自己定义的函数

**库函数：**是由编译系统提供的已设计好的函数，用户只需调用而无需去实现它。前几章用过的 `scanf()`， `printf()`， `getchar()`， `putchar()`， `fmod()`， `sqrt()`等都是库函数。

**用户自定义函数：**由程序员自己定义和设计的函数。需要程序员自己来编写实现函数功能的代码。

## 8.2 函数定义的一般形式

### 一、无参函数的定义形式

```
类型标识符  函数名 ( )  
{  
    说明部分  
    语句  
}
```

无参数传递

例如：

```
void print_message() //void表示无类型  
{  
    printf("\n Hello!");  
}
```



## 二、有参函数的定义形式

类型标识符 函数名（形式参数表列）

{

说明部分  
语句

}

有参数传递

例如：

```
int max (int x, int y) /*求x和y二者中大者， x， y为形参*/  
{  int z;             /*函数体中变量的说明*/  
    z=x>y?x:y;  
    return (z);        /*将z的值作为函数返回值*/  
}
```

### 三、空函数的定义形式

```
类型标识符  函数名( )  
{  
}
```

例如: **void dummy()**  
**{ }**

功能: 调用后什么也不做。

用处: 建立程序结构, 在需要时补充功能。

## 四、对形参说明的传统方式

上面讲的形式参数表的说明形式是新版C语言表示形式（现代方式），即形式参数说明是类型和参数在一起说明。传统的形式参数说明是类型和参数分别说明。

如:按传统方式说明形参

```
int max(x,y)
int x,y;
{
    .....
}
```

按现代方式说明形参

```
int max(int x,int y)
{
    .....
}
```

注意这里是逗号

这两种方式都可以使用，但推荐使用现代方式。

## 8.3 函数参数和函数的值

### 一、形式参数和实际参数

**形式参数：** 在定义函数时函数名后面小括弧中的变量名，简称形参。

**实际参数：** 在调用函数时函数名后面小括弧中的表达式，简称实参。

```
main( )  
{  
    int a,b,c;  
    scanf ("%d", %d, &a, &b);  
    c=max(a,b);  
    printf("Max is %d",c);  
}
```

**实参表**

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

**形参表**

## 上例中形参与实参、函数名与返回值之间的关系：

**c=max(a, b);**

-----  
**max(int x,int y)**

**{ .....**

**returu (z);**

**}**

把函数的  
结果赋给  
函数名

**实参：**在运行时  
把值传给函数。

**形参：**通知系统  
要预留内存位置。

## 关于参数的几点说明:

- ① 形参调用前不占内存单元，调用时占用，调用后释放；
- ② 形参是函数的内部变量，只在函数内部才有意义；
- ③ 对每个形参必须指明其名字和数据类型；

主函数

大宋平民(宋江,李逵)

子函数

梁山英雄(及时雨,黑旋风)

=>

我不认识你!

<=

我也不认识你!



`int songjiang,likui`



`int songjiang,likui`

- ④ 实参可以是常量、变量或表达式，并且必须有确定的值；
- ⑤ 实参个数、类型必须与对应的形参一致；
- ⑥ 实参对形参的数据传递是值传递，即单向传递，只由实参传递给形参，反之不可。调用结束后，只有形参单元被释放，实参单元中的值不变。

## 二、函数的返回值

1. 定义：通过函数调用使主调函数得到一个确定的值，称为函数的返回值。

例如： `c=max(3, 5);`

此时函数的返回值是5，因此`c=5`

### 2. 函数的返回值语句 return

函数的返回值是通过 `return` 语句获得的。  
`return` 语句将被调用函数的一个确定的值带回主调函数中去。`return` 语句的一般形式：

`return(函数返回值);`


`return 函数返回值;`

“函数返回值”是有确定值的常量、变量或表达式。



## 说明:

(1) `return` 后面的值可以是一个表达式，例如：

```
z=x>y? x:y;  
return(z); }  return (x>y? x:y);
```

(2) 一个函数中可以有多条`return`语句，但是一次函数执行只能执行其中的一个。当执行到某个`return`语句时，则终止函数执行，并带回函数值。

(3) 若函数体内没有`return`语句，就一直将函数执行完，再返回调用函数，有一个不确定的值带回。

(4) `return`后面可以无“返回值”（即 **`return ;`**），则该`return`语句只起到终止函数执行，返回主调函数的作用。

### 三、函数值的类型

函数定义时说明的函数的类型(即函数值的类型)，应该与函数返回值的类型一致。

说明：

- (1)凡不加类型说明的函数，一律自动按整型处理。
- (2)如果函数类型和return语句的类型不一致，以函数类型为准。对数值型数据，可以自动进行类型转换。即函数类型决定返回值的类型。
- (3)如果函数没有返回值，可以将函数定义为“无类型” void（或称“空类型”）。

例如： `void print_star(){.....}`

## 8.4 函数的调用

### 一、函数调用的一般形式

有参数函数：

函数名（实际参数表）

无参数函数：

函数名（）

#### 说明：

- (1) 实际参数表中的多个实参间用逗号隔开，实参可以是常量、变量或表达式；
- (2) 实参与形参个数相等，类型一致，按顺序对应，一一传递数据；
- (3) 实参表求值的顺序与系统有关。

## 二、函数调用的具体方式

### 1. 把函数调用作为一个语句

一般形式： 函数名（实际参数表）；

使用情况： 常用于调用一个可以忽略返回值或没有返回值的函数。

如： 对scanf函数和printf函数的调用。

### 2. 函数调用出现在表达式（函数表达式）中

一般形式： 变量名=函数表达式

使用情况： 用于调用带返回值的函数，函数的返回值将参加表达式的运算。

如： `a = 3 + max(num1, num2);`

**注意：** 无返回值函数的调用，不能出现在表达式中。

## 三、对被调用函数的声明

### 1. 函数被调用的前提条件

被调用函数必须是已存在的函数，如用户自定义函数或库函数；而且必须事先声明。

同变量一样，函数的调用也应该遵循“先说明，后使用”的原则。

## 2. 用户自定义函数的函数声明

如果调用用户自定义函数，而且主调函数和被调用函数在同一个文件中，应该在主调函数中对被调函数进行声明。其声明格式如下：

一般形式：

类型标识符 函数名(类型1 形参1, 类型2 形参2, .....);

或 类型标识符 函数名(类型1, 类型2, .....);

在C语言中，以上的函数声明称为函数原型。

对被调用函数的声明

```
main()
```

```
{
```

```
float add(float x,float y);
```

```
float a, b, c;
```

```
scanf(" %f, %f", &a, &b);
```

```
c=add(a,b);
```

```
printf(" sum is %f ", c );
```

```
}
```

```
float add (float x,float y) /*定义add函数*/
```

```
{
```

```
float z;
```

```
z=x+y;
```

```
return (z);
```

```
}
```

作为表达式被调用



### 3. 库函数的函数声明

如果使用库函数，需要在文件的开头用**#include**命令将需要的库函数包含到文件中。

*现在我们清楚了，为什么在使用库函数之前必须包含相应的头文件？那是因为对这些库函数的原型说明全部都写在对应的头文件里了。*

**stdio.h**

.....

**int    \_Cdecl printf(const char \*\_\_format, ...);**

**int    \_Cdecl scanf(const char \*\_\_format, ...);**

**int    \_Cdecl getc(FILE \*\_\_fp);**

**int    \_Cdecl putc(const int \_\_c, FILE \*\_\_fp);**

.....



我们现在用到的头文件有：

```
# include “stdio.h”      /*调用输入输出函数*/  
# include “math.h”       /*调用数学函数*/  
# include “string.h”     /*调用字符，字符串函数*/
```

在本书的附录C中列出了C语言可以使用的标准库函数，大约有75个。请在阅读时注意形参的要求、功能及返回值。

## 4. 函数声明和函数定义的区别

**函数声明**的作用是把函数的名字、函数类型以及形参的类型、个数和顺序通知编译系统，以便在调用该函数时系统按此进行对照检查。

*旧版C语言函数说明只是对函数名及其返回类型的说明。如：float max();未进行全面检查，新版C语言兼容这种用法，但不提倡使用。*

因此，下列函数说明都是合法的：

float max(int x, int y); float max(int, int); float max();

**函数定义**是指对函数功能的确立，包括指定函数值类型、函数名、形参及其类型以及函数体等，它是一个完整的、独立的函数单位。

## 5. 可省略函数声明的三种情况

(1)函数的返回值为整型或字符型时，可以不进行函数声明，系统自动按整型处理。

```
main()  
{ int a,b,c;  
  scanf ("%d,%d",&a,&b);  
  c=max(a,b);  
  printf("Max is %d",c);  
}
```

无函数声明

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

整型函数

(2)函数定义在主调函数之前，可以不进行函数声明。

```
float add (float x, float y)
```

```
{
```

```
    float z;
```

```
    z=x+y;
```

```
    return (z);
```

```
}
```

```
main()
```

```
{
```

```
    float a, b;
```

```
    scanf(" %f, %f, "&a, &b);
```

```
    printf(" sum is %f ", add(a, b) );
```

```
}
```

被调函数在主调函数之前

主调函数在被调函数之后

(3)如果已在所有函数定义之前，在文件的开头，在函数的外部已说明了函数类型，则在各个主调函数中不必对所调用的函数再做声明。

```
char letter( char,char);
```

```
float f(float,float );
```

```
int i(float,float);
```

```
main( )
```

```
{
```

```
.....
```

```
}
```

```
char letter (char c1,char c2)
```

```
{.....}
```

```
float f(float x,float y)
```

```
{.....}
```

```
int i(float,float)
```

```
{.....}
```

在所有函数之前  
说明函数类型

此处不必声明

定义函数  
letter、f和i

## 8.5 数组作为函数的参数

- 数组元素可以做函数的实参
- 数组名可以做函数的参数
- 多维数组可以做函数参数

### 一、数组元素可以做函数的实参

数组元素可以象变量一样作为表达式的组成部分，因此，数组元素可以做函数的实参，单向传递给形参。

例：有两个数组a，b，各有10个元素。将它们逐个对应相比，如果a数组中的元素大于b数组中相应元素的数目多于b数组中的元素大于a数组中相应元素的数目，则认为a数组大于b数组，并分别统计出两个数组相应元素大于，小于和等于的个数。

程序设计：

函数large(x, y)：比较两个数组元素的大小。

$$\text{返回值 flag} = \begin{cases} 1 & \text{当 } x > y \\ 0 & \text{当 } x = y \\ -1 & \text{当 } x < y \end{cases}$$

主函数：输入两个数组，调用large函数比较，计数，输出统计结果。

程序如下：

```
void main ( )
```

```
{  int a[10], b[10], i, l=0, m=0, n=0;
```

```
int large (int x ,int y);
```

---

```
    printf (“enter array a: \n”);
```

```
    for ( i=0; i<10; i++) scanf(“%d”, &a[i]);
```

```
    printf (“ \n”);
```

```
    printf (“enter array b: \n”);
```

```
    for ( i=0;i<10; i++) scanf(“%d”, &b[i]);
```

```
    printf (“ \n”);
```

---

```
for (i=0, i<10, i++)
```

```
{  if (large (a[i], b[i]) == 1) l++;
```

```
    else if (large (a [i], b[i]) == 0) m++;
```

```
    else k++;
```

```
}
```



```
printf(">:%d \n =: %d \n <: %d\n", l, m, n);  
    if (l>n)    printf("a>b\n");  
    else if (l<n) printf("a<b\n");  
    else        printf("a=b\n");  
}
```

*/\*定义large函数\*/*

```
int large (int x ,int y)
```

```
{  int flag;  
    if (x>y) flag=1;  
    else if (x<y) flag= -1;  
    else flag = 0;  
    return (flag);  
}
```

运行输入:

enter array a:

1 3 5 7 9 8 6 4 2 0

enter array b:

5 3 8 9 -1 -3 5 6 0 4

运行结果:

>: 4

=: 1

<: 5

a < b

## 二、数组名可以做函数的参数

由于数组名代表的就是数组在内存中存放区域的首地址。把数组名作为函数参数来实现大量数据的传递是一个非常好的数据传递方法。

### 数组名可以做函数参数的具体方法：

- (1)在主调函数和被调函数中分别定义数组；
- (2)实参数组和形参数组类型应一致；
- (3)实参数组和形参数组大小不一定一致，形参数组可以不指定大小（这里指一维数组）。

例：用选择法对数组中10个整数按由小到大排序。

函数**sort(array[], n)**：对数组元素按由小到大排序。

主程序：输入**array**数组，调用 **sort** 函数排序，输出排序后的**array**数组。

```
void main ( )  
{void sort (int array[ ], int n);  
  int a[10], i;  
  printf("enter array: \n");  
  for (i=0; i<10; i++)  
    scanf ("%d",&a[i]);  
  sort(a, 10);  
  printf("the sorted array: \n");  
  for (i=0; i<10; i++)  
    printf (" %d ", a[i]);  
}
```

由于地址传递，  
实参数组 **a** 改变

```
void sort (int array[ ], int n)
```

```
{  int i,j, k, t;
```

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

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

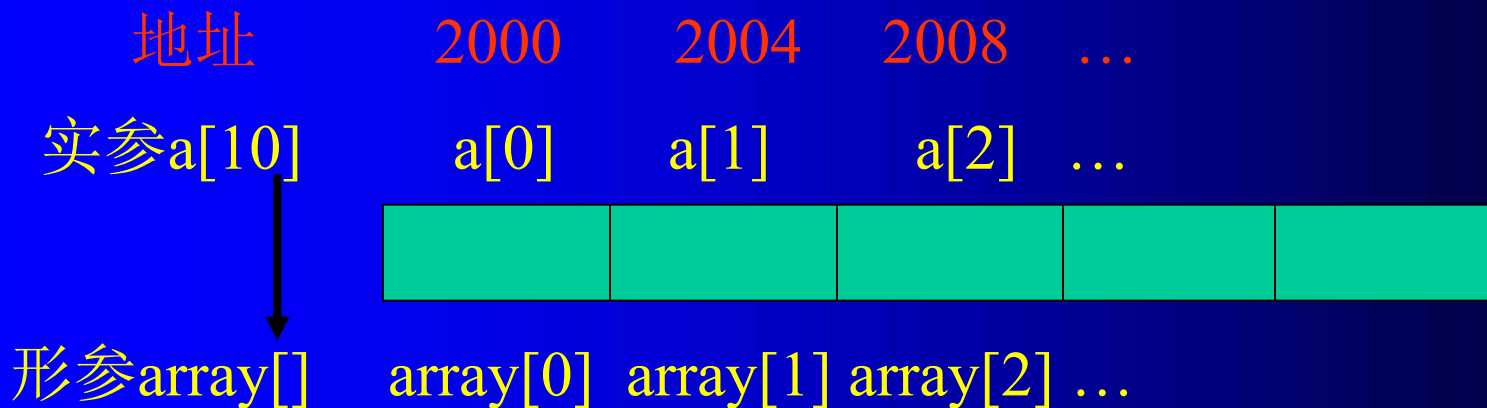
```
        if (array[j]<array[i])
```

```
            {t=array[j]; array[j]=array[i]; array[i]=t;}
```

```
    }
```

```
}
```

这是数组作形参时常使用的技巧



说明：数组名做函数的实参和形参时不是“值传递”，而是“地址传递”。地址传递的结果使得形参与实参共用同一个数组。因此，形参数组各元素的值如果发生变化会使实参数组各元素的值发生同样的变化。

### 数组名做参数的好处：

- (1) 由于只需复制一个地址值，而无须复制全部需要处理的数据，因此节约存储空间并提高效率。
- (2) 由于共用数组，主调函数和被调函数是在相同的内存区域上对数据进行操作，因此可以实现数据的同步更新。

### 三、多维数组可以做函数的参数

- 多维数组元素可以做实参
- 多维数组名可以做参数

说明：

(1)形参数组定义时可以指定或省略第一维的大小。

如：`int array [3][10];` 或 `int array [ ][10];`

但 `int array [3][ ];` ✗ 和 `int array [ ][ ];` ✗

(2)实参数组可以大于形参数组。

如：实参数组定义为：`int array [5][10];`

形参数组定义为：`int array [3][10];`

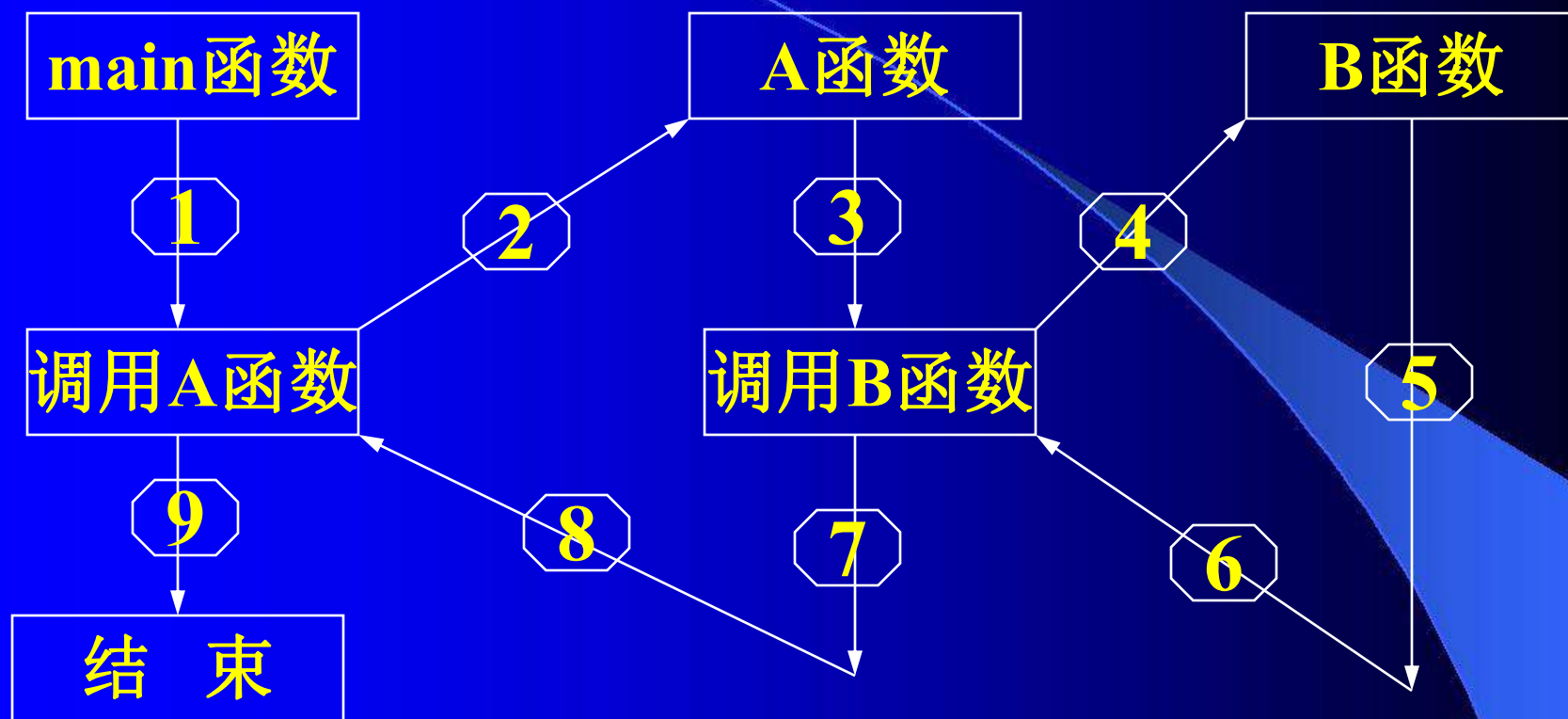
这时形参数组只取实参数组的一部分，其余部分不起作用。

## 8.6 函数的嵌套调用

**嵌套调用：**在调用一个函数的过程中，该函数又调用其它函数。

~~**嵌套定义：**在定义一个函数时，该函数体内包含另一个函数的定义。~~

C语言不能嵌套定义，但可以嵌套调用。在调用一个函数的过程中，又调用另一个函数。



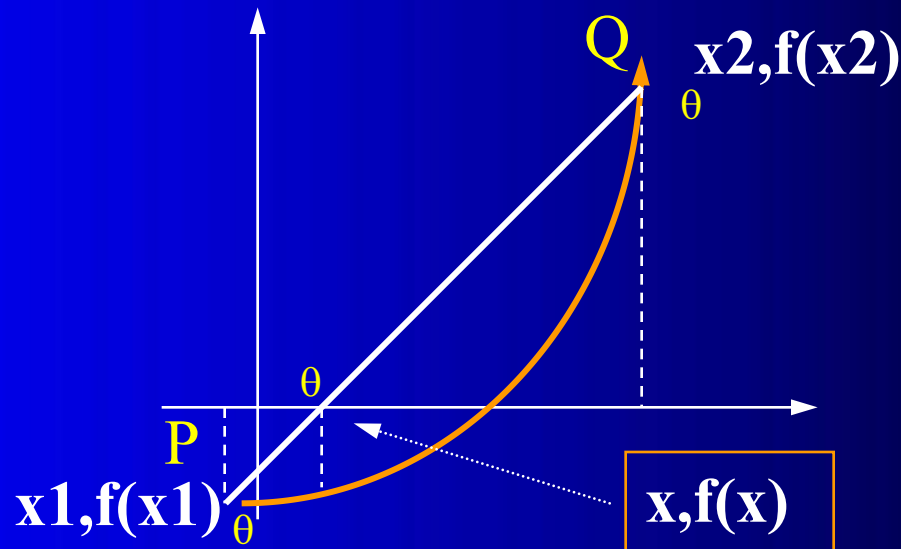


## [例8.6] 用弦截法求方程的根

本例较复杂，重点是了解如何将复杂的问题分解，然后再整合，最终解决问题。

数学函数（公式）为： $y = f(x) = x^3 - 5x + 16x - 80 = 0$

问题可转化为：求曲线与x轴交点的 x 坐标



# 1. 解题方法

(1)在曲线上取两点P和 Q 坐标为 $(x_1, f(x_1))$ 和 $(x_2, f(x_2))$ ，如果 $f(x_1)$ 和 $f(x_2)$ 符号相反，则 $(x_1, x_2)$ 区间内必有一根，反之，改变 $x_1$ 和 $x_2$ 的值，直到 $f(x_1)$ 和 $f(x_2)$ 异号为止。

(2)求连接P和 Q 点的线段与x轴的交点x。

$$x = (x_1 * f(x_2) - x_2 * f(x_1)) / (f(x_2) - f(x_1))$$

(3)如果  $f(x)$  和  $f(x_1)$  符号相同， $(x, x_2)$ 区间内必有一根，此时将x作为新的 $x_1$ 。如果  $f(x)$  和  $f(x_2)$  符号相同， $(x_1, x)$ 区间内必有一根，此时将x作为新的 $x_2$ 。

(4)重复步骤 (2)和(3)，不断缩小 $(x_1, x_2)$ 的距离，同时  $f(x)$ 越来越趋于0，直到  $|f(x)| < \varepsilon$ 。

## 2. 程序设计:

函数  $f(x)$  :

求  $x^3 - 5x^2 + 16x - 80$  的值

函数  $xpoint(x_1, x_2)$  :

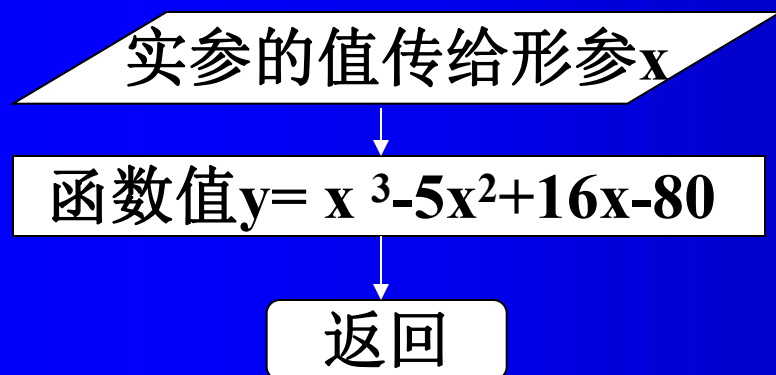
求  $(x_1, f(x_1))$  和  $(x_2, f(x_2))$  的连线与  $x$  轴的交点  $x$ 。

函数  $root(x_1, x_2)$  :

求  $(x_1, x_2)$  区间内的根。

# 程序流程图:

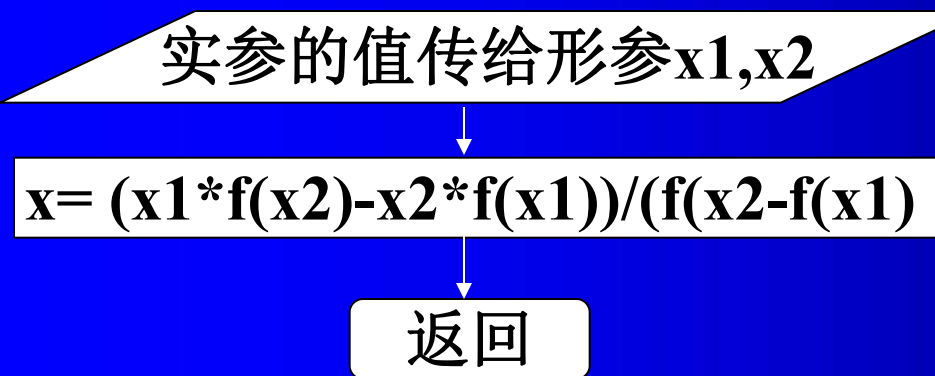
## 求f(x)函数流程图:



/\*定义f函数的源程序\*/

```
double f(double x)
{
    double y;
    y=x*(x*(x-5.0)+16.0)-80.0;
    return (y);
}
```

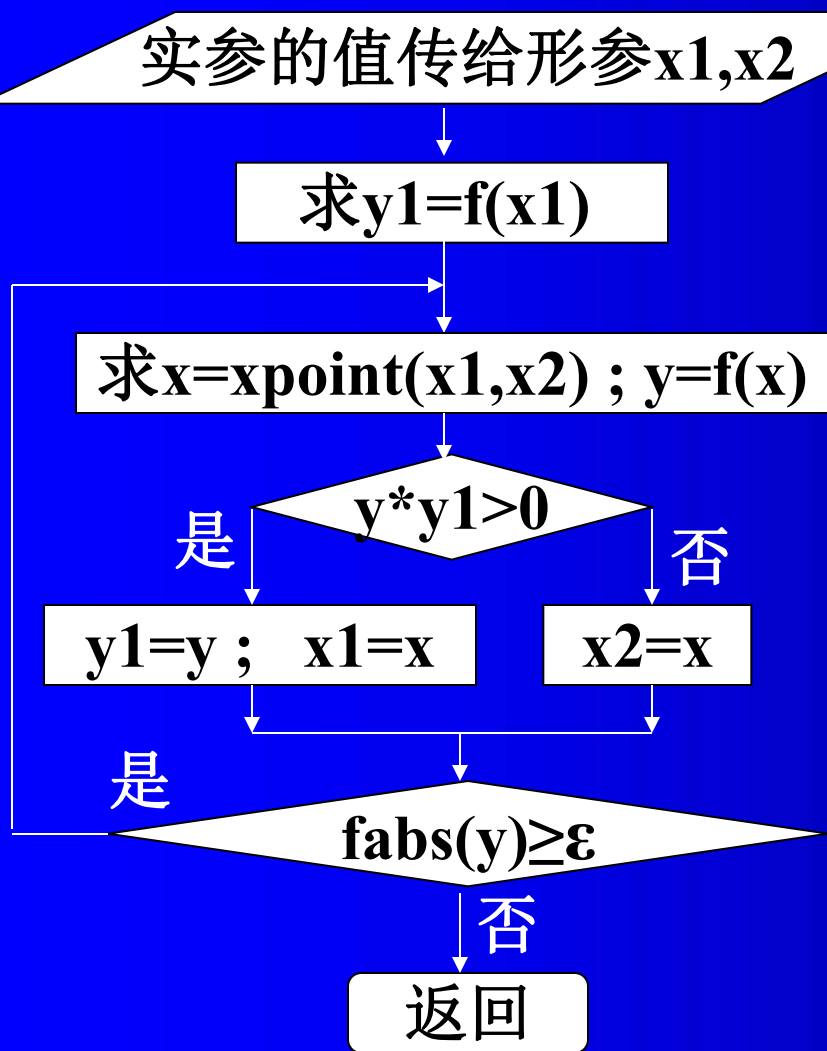
## 求xpoint(x1,x2)函数流程图:



/\*定义xpoint函数的源程序\*/

```
double xpoint (double x1, double x2)
{
    double x;
    x=(x1*f(x2)-x2*f(x1))/(f(x2)-f(x1));
    return (x);
}
```

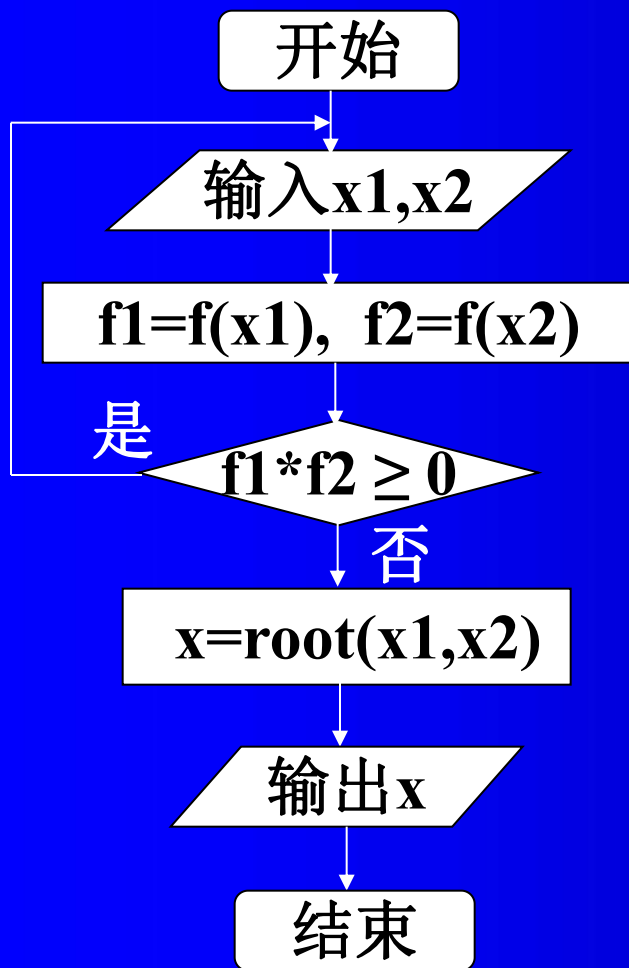
## 求root(x1,x2)函数流程图:



/\*定义root函数的源程序\*/

```
double root (double x1, double x2)
{double x, y, y1;
  y1=f (x1);
  do
  { x=xpoint(x1, x2);
    y=f(x);
    if (y*y1>0)
      {y1=y; x1=x; }
    else x2=x;
  }while(fabs(y)>=0.0001 );
  return (x);
}
```

## 主函数流程图:



```
#include "math.h"
```

```
void main ( )
```

```
{double f(double); //函数声明
```

```
double xpoint (double, double); //函数声明
```

```
double root (double, double); //函数声明
```

```
double x1, x2, f1, f2, x;
```

```
do
```

```
{ printf (" input x1, x2:\n");
```

```
scanf ( " %f, %f ", &x1, &x2);
```

```
f1=f(x1); f2=f(x2);
```

```
}while (f1*f2>=0);
```

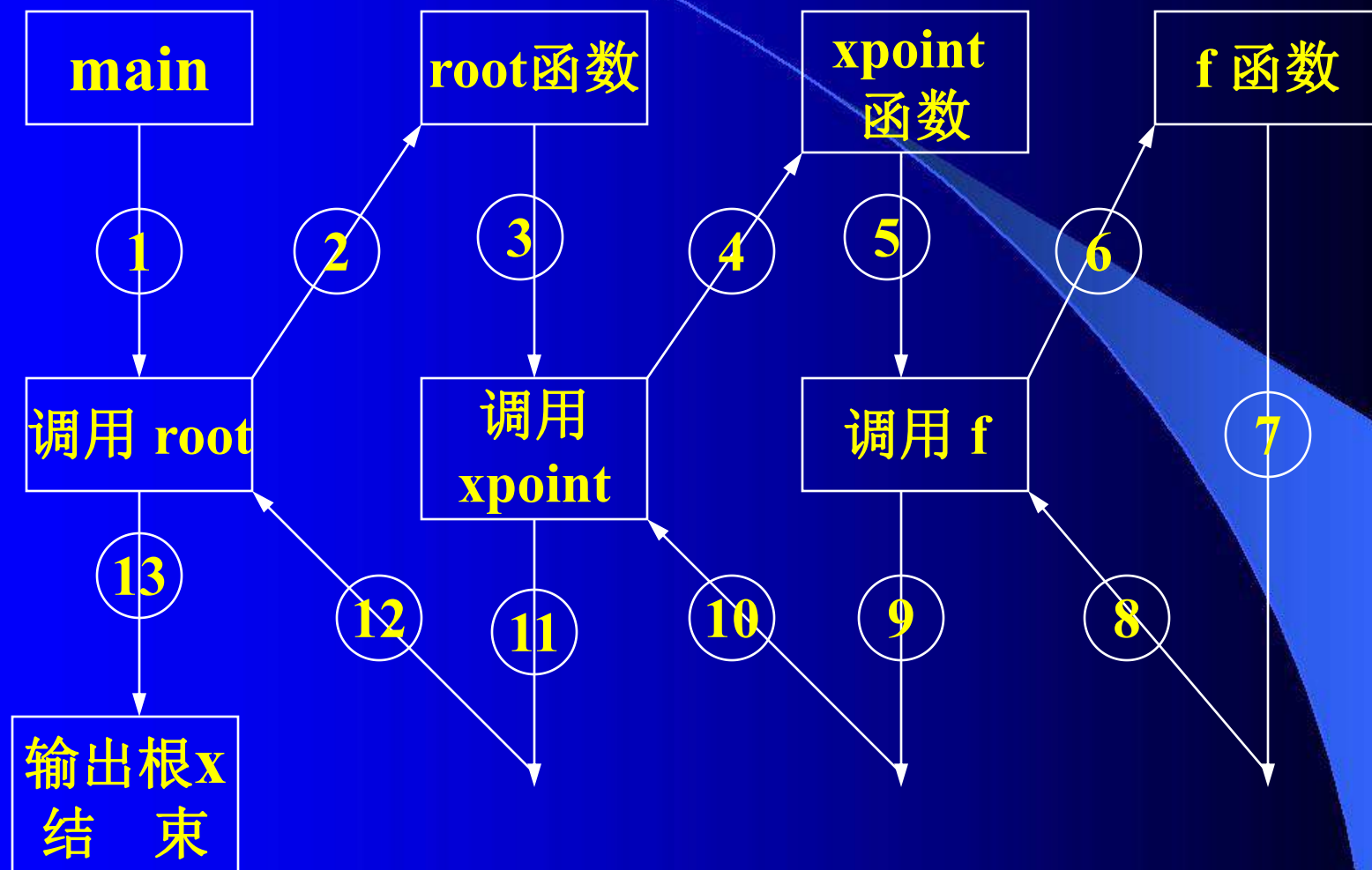
```
x=root (x1, x2);
```

```
printf ("A root is %8.4f",x);
```

```
}
```

注意：因用到库函数fabs，故应在文件开头加上：`#include "math.h"`

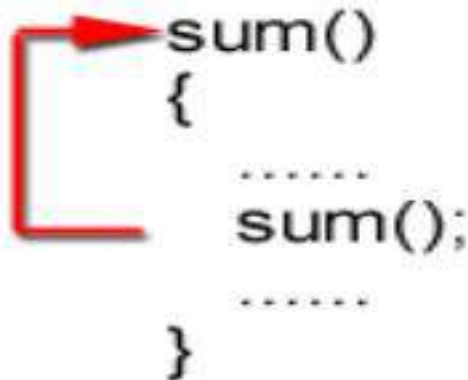
## 上题函数的嵌套调用关系:



## 8.7 函数的递归调用

**定义：** 在一个函数的定义过程中直接或间接地调用该函数本身。

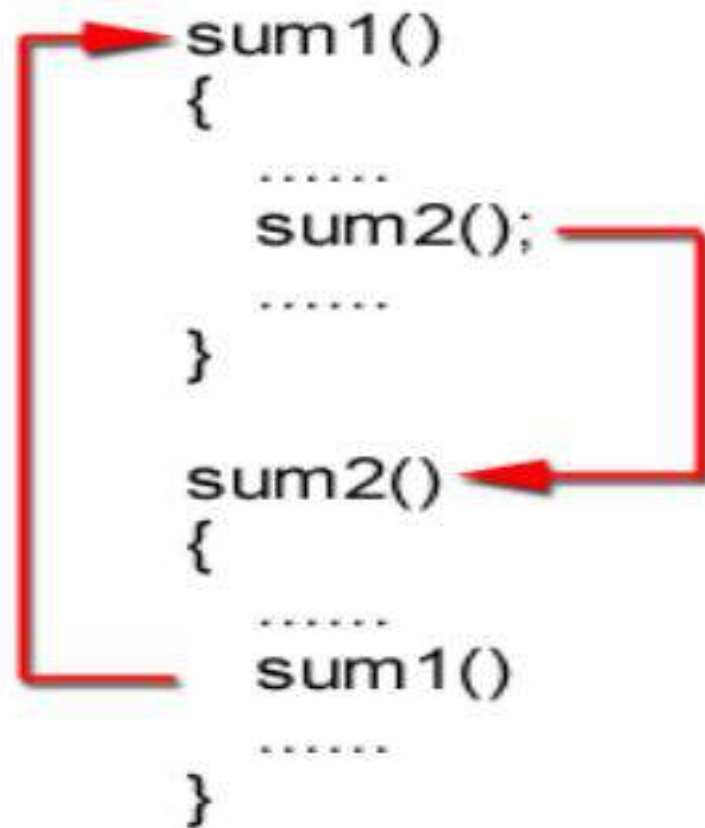
直接递归调用



```
sum()
{
    .....
    sum();
    .....
}
```

The diagram illustrates a direct recursive call. A red arrow originates from the `sum();` line within the function's body and points back to the `sum()` function name at the top of the block, indicating a self-call.

间接递归调用



```
sum1()
{
    .....
    sum2();
    .....
}

sum2()
{
    .....
    sum1();
    .....
}
```

The diagram illustrates an indirect recursive call. It shows two functions: `sum1()` and `sum2()`. A red arrow points from `sum2();` in the `sum1()` function to the `sum2()` function definition below. Another red arrow points from `sum1();` in the `sum2()` function back to the `sum1()` function definition above, forming a cycle of mutual calls.



说明：不应出现无终止的递归调用，因此，应该  
给定一个限制递归次数的条件。

递归调用对应的一般算法：

$$f(x)=\begin{cases} \text{终了公式} \\ \text{递归公式} \end{cases}$$

递归函数的执行过程：

递归调用：记住本次现场，递归调用。

终了调用：返回上次调用现场。

## 例：用递归法求n!

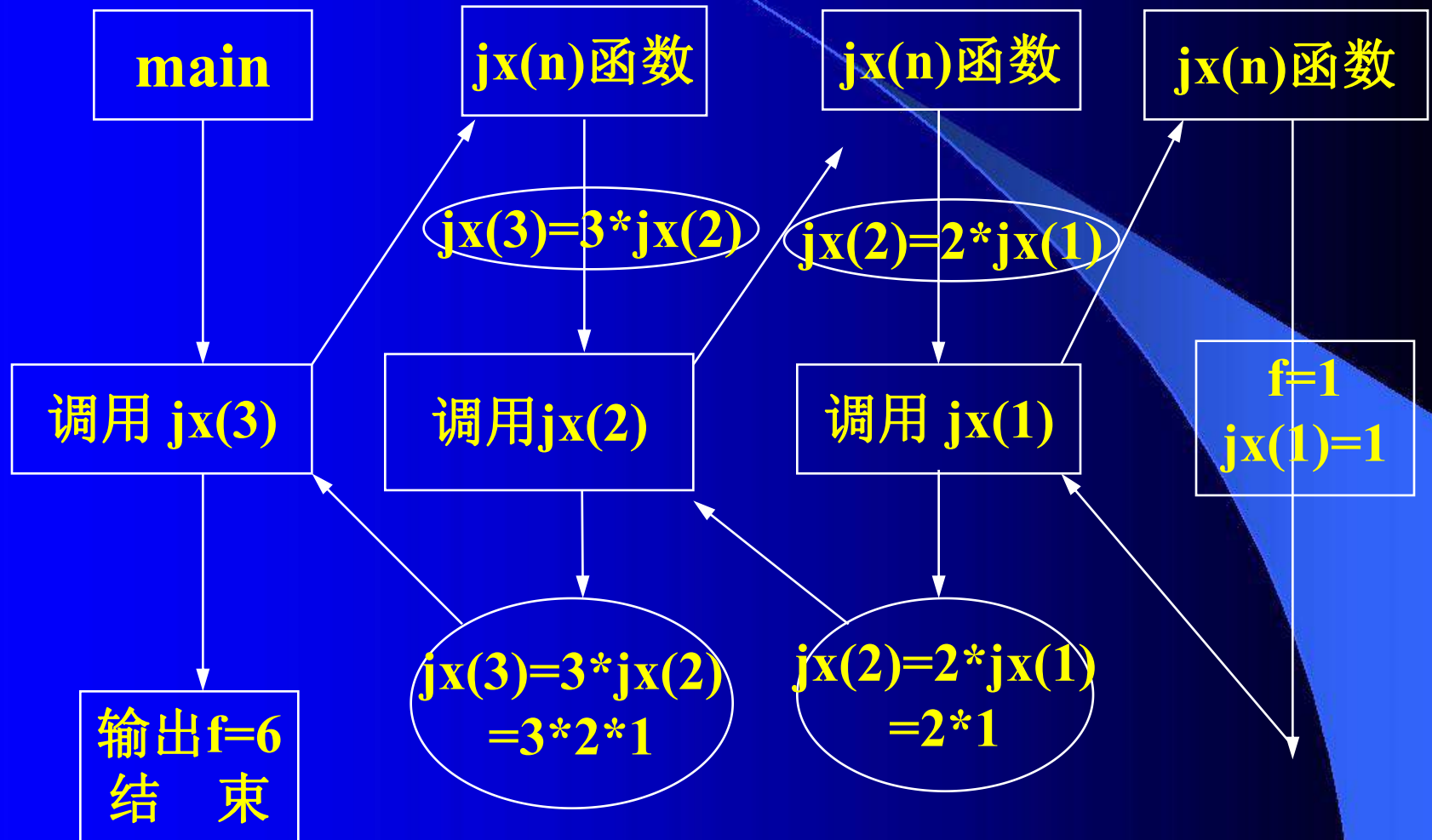
算法：

$$f(n) = \begin{cases} 1 & (n=1) \\ f(n-1)*n & (n>1) \end{cases}$$

```
main()
{
    long jx(int);
    int n;
    long f;
    printf("Input n :");
    scanf("%d",&n);
    f= jx(n);
    printf("\n n!=%d",f);
}
```

```
long jx(int n)
{
    long f;
    if(n==1) f=1;
    else f=n*jx(n-1);
    return f;
}
//n!=n*jx(n-1)
//jx(n1)= (n-1)*jx(n-2)
// ...
```

## 上题函数的嵌套调用关系:



## 8.8 局部变量和全局变量

### 一、局部变量

定义：在函数内部或复合语句内部定义的变量，称作**局部变量**。

作用域：函数内或复合语句(Nested Block)内。

注意：

- (1)主函数 `main` 定义的变量只在主函数中有效，  
主函数不能使用其它函数定义的变量。
- (2)不同函数中的同名变量互不影响。
- (3)形参也是局部变量。

```
例1: void function1(void)
      void function2(void);
      main()
      {
          int m=1000;
          function2();
          printf(“%d\n”,m );
      } /*第三次输出*/
      void function2(void)
      {
          int m=100;
          function1();
          printf(“%d\n”,m );
      } /*第二次输出*/
      void function1(void)
      {
          int m=10;
          printf(“%d\n”,m );
      } /*第一次输出*/
```

运行结果: 10  
100  
1000

## 例2:

```
main()
```

```
{ int a=20,b=10;
```

```
....
```

```
{ int a=0;
```

```
  int c=a+b;
```

```
...
```

```
}
```

```
  b=a;
```

```
}
```

c在此范围内有效

a、b在此范围内有效

运行结果: a=20  
b=20  
c=10

## 二、全局变量

定义：在函数外部定义的变量称作**全局变量**（也称外部变量）。

作用域：可以为本文件中的所有函数公用。

注意：

- (1)从定义变量的位置开始到本文件结束，这段程序中的函数可直接使用外部变量。
- (2)如果在定义点之前的函数想使用外部变量，则应该在该函数中用关键字 `extern` 作“外部变量”说明。
- (3)如果在同一源文件中，全局变量和局部变量同名，则在局部变量的作用范围内，全局变量不起作用。

**例3:**

**#include "c.h"**

**double a=20,b=10,c=12;**

**main()**

**{**

**....**

**s=0.5\*(a+b+c);**

**....**

**}**

**int q=3.1415926/180;**

**double sub1(){}  
double sub2(){}  
}**

**例4:**

**int a=200,b=100;**

**main()**

**{ int a=20,b=10;**

**....**

**{ int a=2,b=1;**

**int c=a+b; }**

**....**

**b=a;**

**}**



## 从程序设计的观点看使用全局变量：

优点：(1)增加了函数间数据联系

同一文件中的一些函数引用全局变量，当某个函数中改变了全局变量的值，其它函数中的全局变量值也随之改变。

(2)函数可以得到多个返回值

缺点：(1)全局变量在程序的全部执行过程中都占用存储单元。

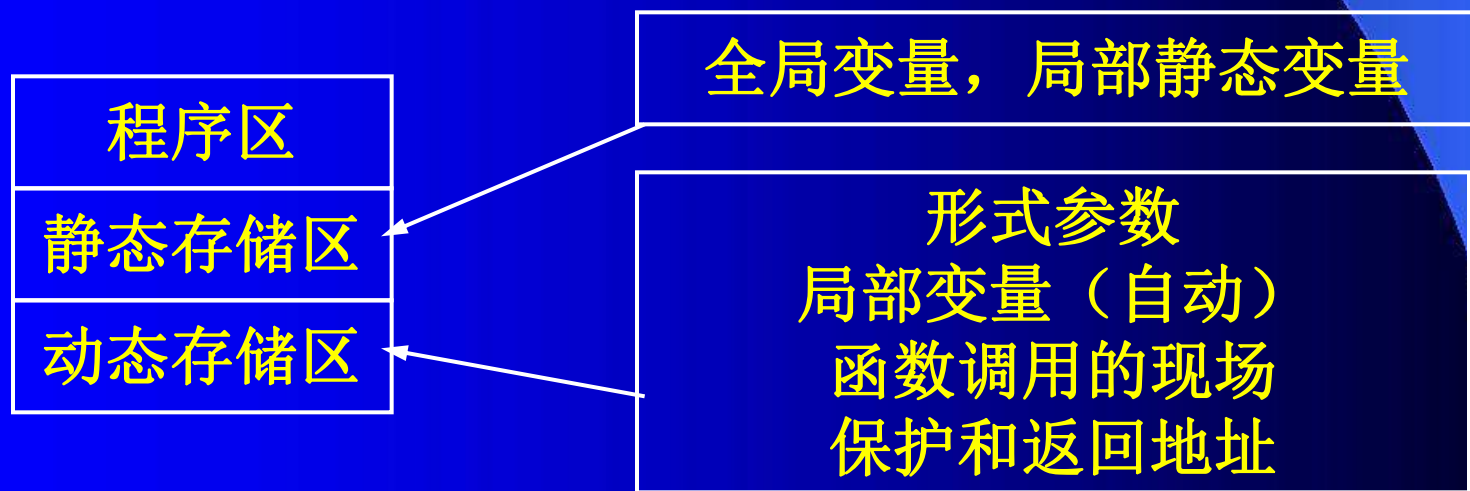
(2)使用全局变量不符合程序设计中要求模块间“强内聚性、弱耦合性”的原则。

(3)使用全局变量过多，会降低程序的可读性和可维护性。

## 8.9 变量的存储类别 (storage class)

### 一、静态存储方式和动态存储方式

- 静态存储方式：程序运行期间分配固定存储空间的方式。
- 动态存储方式：程序运行期间根据需要进行动态的分配存储空间的方式。



## 二、静态存储变量和动态存储变量


- 静态存储变量：用静态存储方式存储的变量。

特点：在静态存储区分配存储单元，整个程序运行期间都不释放。

- 动态存储变量：用动态存储方式存储的变量。

特点：函数开始调用时为变量分配存储空间，函数结束时释放这些空间。一个程序两次调用同一函数，其中同一个局部变量的内存地址可能不同。

### 三、变量的属性及其定义

C语言变量的属性：数据类型  
存储类型

数据类型：整型，实型，字符型.....

存储类型：数据在内存中的存储方式，即静态存储方式和动态存储方式。通过存储分类符来表示。

变量的按存储类别分类：(storage class)

**automatic variables**(自动变量)

**register variables** (寄存器变量)

**static variables** (静态变量)

**extern variables** (外部变量)

## 定义变量的一般形式:

存储分类符 类型标识符 变量名;

```
如: auto int a;  
     static int b;  
     register int d;
```

上述变量的属性: 作用域(Scope), 生存期(Longevity), 可见性(Visibility)

作用域: 变量可以使用的程序范围 (活动的)

生存期: 变量在程序执行的一段时间内保持存活的状态 (生存的)

可见性: 变量可从内存中存取的特性

## 四、变量的存储类型

### 1. **automatic** variables(自动变量)

■自动变量在它们所使用的函数中声明，随函数的调用而生成，而当退出函数时自动终结，因而得名自动变量。自动变量是它们所在函数的局部变量，因而被称为局部变量或内部变量。

■函数中定义的变量如果没有声明为**auto**，则隐含指定为自动变量，数据存储在动态存储区中，属于动态存储变量。程序中大多数变量属于自动变量。

自动变量定义为**auto int a;**



说明：(1)在一个函数内如果局部变量不作存储类型说明，均为自动变量；

如： `int b, c=3` 等价于 `auto int b, c=3;`

(2)形式参数缺省存储类型是`auto`，但不能将`auto`加在形参说明之前。

如： `int max(auto int x, auto int y){.....}` ✗

## 2. `static` variables (静态变量)

■ 静态变量的生存期为从其定义开始到整个程序结束。

■ `static`可用于说明局部静态变量和外部静态变量。

## 局部静态变量

- (1) 定义：在函数内部定义
- (2) 作用域：定义它的函数内部。
- (3) 生存期：存放在静态存储区，整个程序运行期间都不释放。

*而自动变量在函数调用结束后即释放。*

- (4) 使用：编译时赋初值，每次调用时不再赋初值，只保留调用结束时变量的值。

*而自动变量调用一次，重新赋值一次。*



例如：打印1! ~5!

```
int fac(int n)
{static int f=1;
  f=f*n;
  return(f)
}
main()
{
  int i;
  for(i=1;i<=5;i++)
    printf("%d!=%d\n",i,fac(i));
}
```

**(5)如果局部静态变量不赋初值，编译时自动赋0。  
而自动变量不赋初值，其值不确定。**

## 外部静态变量

- (1)定义：在所有的函数之外定义。
- (2)作用域：可以被本程序文件中所有的函数使用，而不能被其它程序文件引用。
- (3)生存期：整个程序运行期间，编译时分配在静态存储区（不论是否加static说明）。
- (4) 静态外部变量与普通外部变量的区别：

外部静态变量只能被它定义时所在的文件使用，而普通外部变量能被其它文件使用。

```
例如：#include "b.cpp" //can use pd_swerling_2
      static int pd_swerling_1; //外部静态变量
      int pd_swerling_2; //全局变量
      void main(){}      .....
```

### 3. **register** variables (寄存器变量)

寄存器：它是CPU中运算器的组成部分，用来暂时存放数据的存储装置。

寄存器变量：直接放置在运算器的寄存器中的变量称为寄存器变量。引入寄存器变量是为了提高“存取”速度。

(1) **register**只适用于说明局部变量；

例如：`register int i;`

定义了*i*是寄存器变量，其数据类型是整型的。

(2) 寄存器变量属于动态存储变量，但并不放在动态存储区中，它放在寄存器中。

(3) 寄存器变量的作用域与自动变量相同。

说明：

(1)只有局部变量（非静态的）和形式参数可以作为寄存器变量。

例如： `register static int i;` ✗

(2)由于计算机的寄存器数目有限，不宜定义太多寄存器变量。不同的系统允许使用的寄存器个数是不同的。

## 4. **external variables** （外部变量）

(1)生存期与作用域：在整个程序运行期间都“存活”和“有效”的变量称为外部变量； 也称为全局变量。

(2)在多个源程序文件组成的程序中

➤如果B文件中引用在A文件中定义的全局变量（该全局变量不是static变量），则需要在B文件中，用关键字extern说明。

也可以写成extern a;

例如：

```
/*文件A.c*/
```

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

```
/*文件B.c*/
```

```
extern int a;  
power()  
{  
    .....  
}
```

➤若在A文件中的全局变量是静态外部变量（**static**），则即使在B文件中用**extern**对其说明，也无法在B文件中使用该全局变量。

例如：

```
/*文件A.c*/  
static int a;  
main()  
{  
    .....  
}
```

```
/*文件B.c*/  
extern int a;  
power()  
{  
    .....  
}
```

无法使用  
变量a

(3) 在同一个源文件中，全局变量的作用域之外的函数中使用该全局变量时，也需用extern对其说明。

例如：

```
extern int a;  
  
power()  
{  
    .....  
}  
  
int a;  
main()  
{  
    .....  
}
```



名称	普通外部变量	局部静态变量	外部静态变量	外部变量
解释	全局变量	静态(当地)局部变量	静态(当地)外部变量	引用
存储类型	无	<b>static</b>	<b>static</b>	<b>extern</b>
声明位置	文件中的所有函数前（可以初始化）	函数中（可初始化）	文件中的所有函数前（可初始化）	文件中的所有函数前（不能初始化）
作用域	本文件; 用 <b>extern</b> 进行变量声明的其它文件	函数	本文件	定义该全局变量的原始文件; 本文件; 用 <b>extern</b> 进行变量声明的其它文件
生存期	整个程序运行期间（全周期）	全周期	全周期	全周期



## 8.10 内部函数和外部函数

### 一、内部函数

定义：如果一个函数只能被本文件中其它函数调用，称为内部函数（又称静态函数）。

格式：`static` 类型标识符 函数名（形参表）

```
例如：static int fun(a, b)  
      {  
          .....  
      }
```

作用：函数的作用域限于所在文件，不同文件中同名函数互不干扰，便于程序的局部化。

## 二、外部函数

定义：如果一个函数允许被其它文件调用，称为外部函数。

格式：**extern** 类型标识符 函数名（形参表）  
或 类型标识符 函数名（形参表）

例如：**extern int fun(a, b)**  
    {  
        .....  
    }

或

**int fun(a, b)**  
    {  
        .....  
    }

通常不加 **static** 标识符的函数都是外部函数。

在需要调用此函数的文件中，一般要用 **extern** 说明所用的函数是外部函数。

# 变量的作用域与生存期

名称	存储类型	声明位置	作用域	生存期
全局变量	无	文件中的所有函数前 (可初始化)	本文件+用extern进行变量声明的其它文件	整个程序运行期间 (全周期)
外部变量	extern	文件中的所有函数前 (不能初始化)	定义该全局变量的原始文件+本文件+用extern进行变量声明的其它文件	全周期
静态外部变量	static	文件中的所有函数前	本文件	全周期
自动变量	无 或auto	函数或复合语句中	函数或复合语句	函数或复合语句结束
寄存器变量	register	函数或复合语句中	函数或复合语句	函数或复合语句结束
静态局部变量	static	函数中	函数	全周期

# 本章作业:

- 1.采用主函数与子函数或递归函数结合的方法计算如下级数:

$$f(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \dots$$

要求计算的精度为 $1 \times 10^{-5}$ , 即前后两次求和结果的差值不超过该值 ( $|S_{n+1} - S_n| \leq 1 \times 10^{-5}$ )

2. 在主函数中输入两个字符串, 写一函数将其连接, 再在主函数中将其输出。

3. 在主函数中输入一个四位数，编写子函数将其各位上的数字拆解出来。在主函数中将各个数字输出，要求每两个数字之间空一个空格。例如输入2008，则应输出“2 0 0 8”。

4. 编写用牛顿迭代法求根的子函数。方程为  $ax^3 + bx^2 + cx + d = 0$ ，系数a、b、c、d的值依次为1、-2、3、-4，由主函数输入。计算在1~2之间的一个实根，求出根后由主函数输出。

## 5.判断对错

- 1) C函数通过函数名只能返回一个值;
- 2) C函数至少要有一个形参;
- 3) 函数可以在main函数前定义;
- 4) 函数可以在main函数之中定义;
- 5) 用户定义的函数至少应被调用一次, 否则就会出现警告信息;
- 6) 任何名字都可用作函数名;
- 7) 只有void类型的函数可以用void作为其形参;
- 8) 全局函数在程序中所有的复合语句和函数中都是起作用的;

- 9) 函数可以调用它自己;
- 10) 没有return语句的函数是非法的;
- 11) 全局变量不能被声明为auto变量;
- 12) 函数原形必须永远放在主调函数的外面;
- 13) 函数的返回值类型缺省为int;
- 14) 在函数原形中的变量类型应该与在函数定义中相同;
- 15) 在将数组传递给函数时, 函数调用必须用不带方括号的数组名。



## 6. 填空

- 1) 函数调用时的参数称为\_\_\_\_\_;
- 2) 在函数内部声明的变量称为\_\_\_\_\_;
- 3) 缺省情况下, C函数的返回值类型为\_\_\_\_\_;
- 4) \_\_\_\_\_指的是变量实际可以使用的时期;
- 5) 调用它自己的函数被称为\_\_\_\_\_函数;
- 6) 如果一个局部变量要在调用回到该函数保持它的值存在, 该局部变量应该声明为\_\_\_\_\_;
- 7) \_\_\_\_\_帮助编译检查实参与形参之间的匹配关系;
- 8) 缺省情况下, 函数类的变量声明具有\_\_\_\_\_存储类型。



# 第十章 指 针(Pointers)

- 掌握指针与指针变量的概念；
- 掌握数组的指针和指针数组的使用；
- 掌握字符串指针和指向字符串的指针变量的使用；
- 掌握指针函数和函数指针的使用；
- 了解指向指针的指针的概念及其使用。

Pointers are undoubtedly one of the most distinct and exciting features of C language. It has added power and flexibility to the language. Although they appear little confusing and difficult to understand for a beginner, they are powerful tool and handy to use once they are mastered.

- 毫无疑问，指针是C语言最独特和令人兴奋的特性之一，它增加了这门语言的能力和灵活性。尽管它们对于初学者来说显得有点令人糊涂和不易理解，但一旦被掌握，他们将成为强有力的工具并变得易于使用。

Pointers offer a number of benefits to the programmers. They include:

- 1) Pointers are more efficient in handling arrays and data tables;
- 2) Pointers can be used to return multiple values from a function via function arguments.
- 3) The use of pointer arrays to character strings results in saving of data storage space in memory.
- 4) Pointers reduce length and complexity of programs.
- 5) They increase the execution speed and thus reduce the program execution time.....

指针对编程者提供了许多好处，包括：

- 1) 在处理数组和数据表方面更有效；
- 2) 被用于通过函数变量从一个函数中返回多个函数值；
- 3) 指向字符串的指针数组的使用可以节省内存中的数组存储空间；
- 4) 降低了程序的长度和复杂性；
- 5) 加快了程序处理速度，减小了执行时间.....

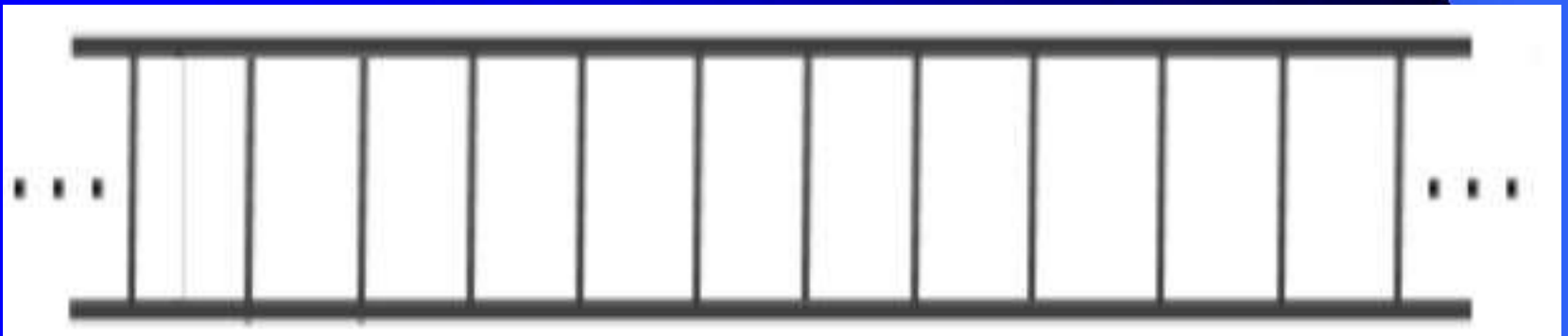
# 预备知识

## 1. 内存的概念(Concept of the Memory)

**内存：** CPU处理的数据都保存在内存中。内存分为一个个单元，这些单元顺序排列，每一个单元有一个称为内存地址的编号，对内存数据的访问都是通过地址进行的。

高级语言用变量等高级概念把内存单元、地址等低级概念掩盖起来，使我们在写程序时不必关心硬件的细节，但内存与地址仍是最基本的概念。

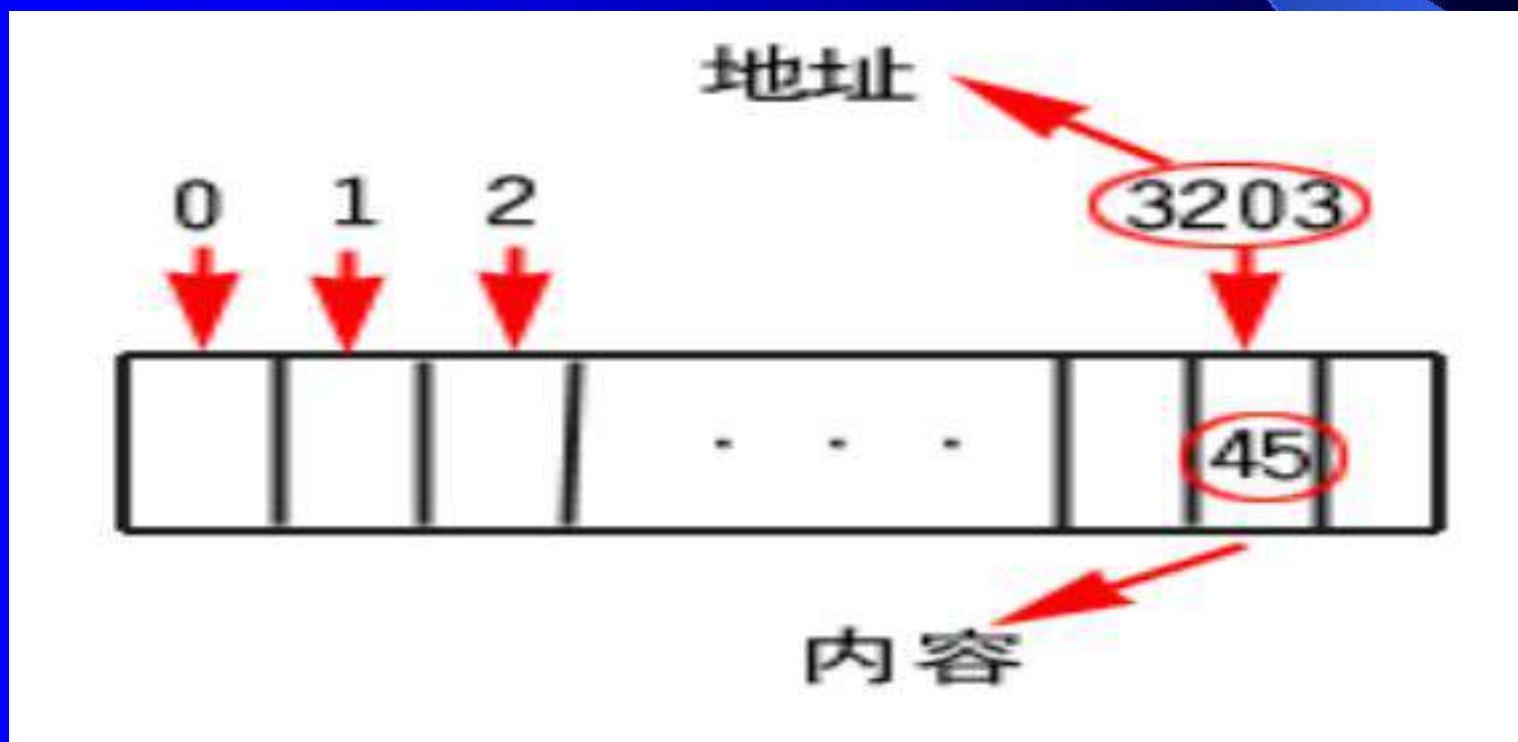
存储单元的最小单位是字节。



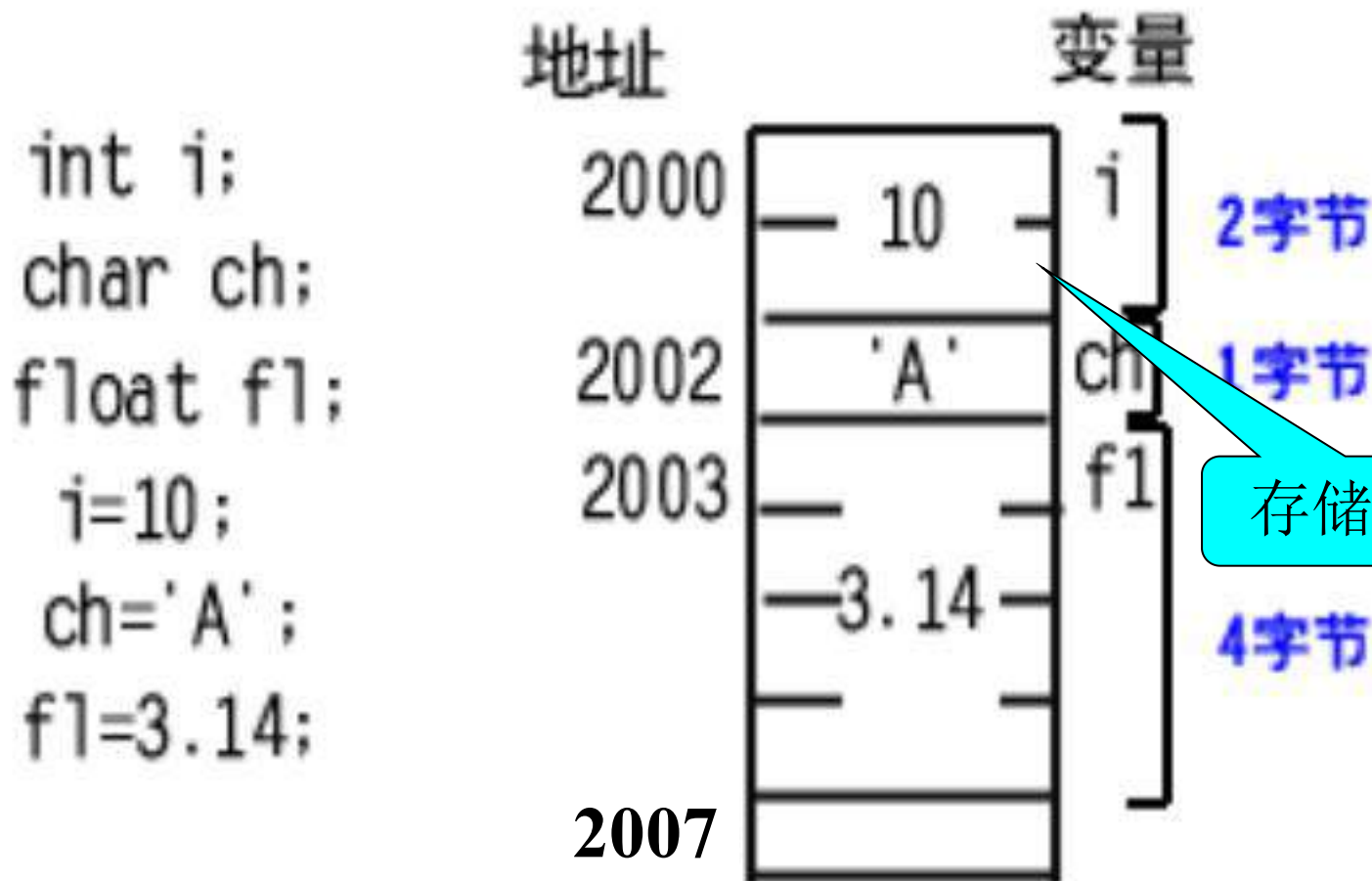
## 2. 地址的概念(The Concept of address)

**地址：**为了对内存中的某个存储单元进行访问，要  
为它编号，这种编号称为内存地址。

通过地址我们就能够访问该地址所标识的存  
储单元。



**变量的地址：**每一个变量在内存中总占用几个连续的字节，开始字节的地址，就是变量的地址。





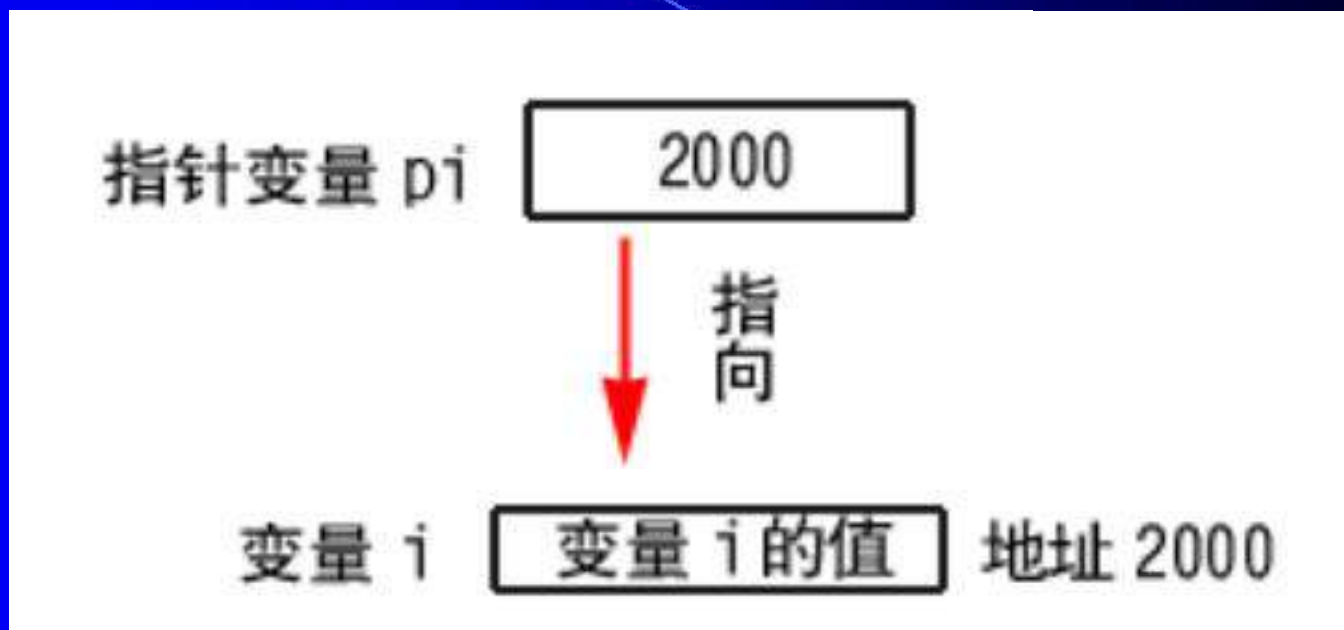
## § 10.1 指针及其相关概念

**指针：**一个变量的地址称为该变量的指针。许多高级语言都把程序对象的地址作为一种数据，称之为地址值或指针值。

**指针变量：**若一个变量专用于存放另一个变量的地址（指针），则前者称为指针变量。换句话说，以地址为值的变量称为指针变量，简称指针。



指针的对象：当把变量的地址存入指针变量后，我们就可以说这个指针指向了该变量。



变量的存取方法：直接存取和间接存取。

- 直接存取：直接根据变量名存取数据。
- 间接存取：通过指针变量存取相应变量的数据。



## § 10.2 变量的指针和指向变量的指针变量

### 一、指针变量的定义 Declaring pointer variables

一般形式:

类型标识符 \*变量名 ;

type \*variables\_name ;

例如: `int *ptr1, *ptr2, i, j, k;`

`double *q, *s, x, y, z;`

`char *ch1, ch2, ch3[10];`

指针变量的类型: 说明该指针所指向的变量的类型。

Declaring the variable type to which the pointer is pointed.

## 在定义指针变量时要注意以下几个问题:

(1) 变量名ptr2前面的“\*”不能省略, 如果写成

```
int *ptr1, ptr2;
```

则ptr2被定义为整型变量, 而非整型指针变量。

ptr2 is a integer variable.

(2) 定义中的“\*”表示所定义的变量是指针变量,

但指针变量名是ptr1、ptr2, 而非\*ptr1、\*ptr2。

‘int \*’ means to declare an integer pointer. The point variable name is ‘ptr1、ptr2’, but not ‘\*ptr1、\*ptr2’.

(3)指针变量只能指向定义时所规定类型的变量。

这个规定的类型称为该指针变量的“基类型”。

**The type of a pointer must match with the type of the variable to which the pointer is pointed. The type is called the ‘basic type’ of the pointer.**

如：上例中ptr1、ptr2只能指向整型变量，不能指向实型或字符型变量。其“基类型”相同，都是整型。

(4)定义指针变量后，并未确定该变量指向何处。

也就是说该变量的值是不确定的。在引用指针变量前必须首先让它指向一个变量，这一点非常重要。

## 二、指针变量的运算

### Operations of the pointer

#### (一) 指针运算符(“&”和 “\*” )Pointer operators

“&”(取地址运算符)：取变量的存储地址

The & operator can be remembered as ‘address of’, its function is to access the *address of a variable*.

例如：&a 是求变量a的地址。

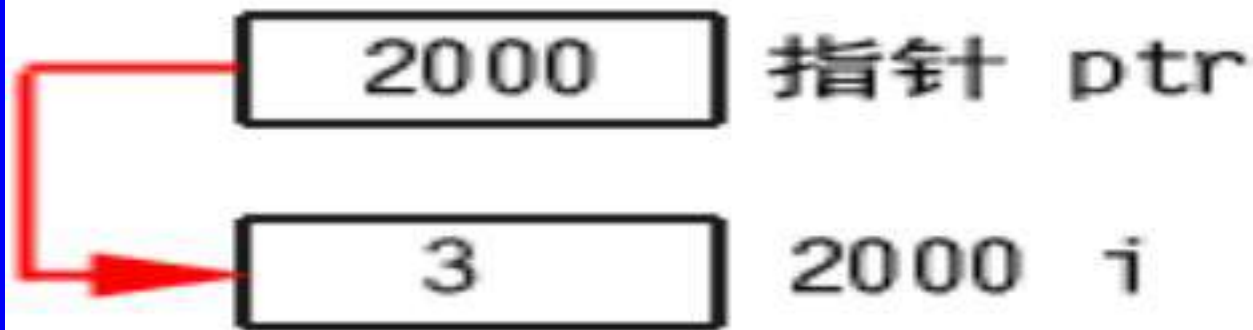
&a is evaluating the address of argument ‘a’

“\*” (取值运算符)：取指针所指向变量的内容

The \* can be remembered as ‘value at address’. its function is to access the *value of a variable* to which the pointer is pointed.

```
int *ptr, i;
```

```
i=3;  
ptr=&i;
```



ptr指向了i变量，\*ptr表示i的值，即3

The pointer 'ptr' was pointed to the argument 'i',  
\* ptr expresses the value of the 'i', thus 3.

我们还可以用这种方法实现对变量的改变：

```
*ptr = 15;
```

2000 指针 ptr

15 2000 i

**\*ptr=15 ; 等价于 i=15 ;**

由此可见：通过指针运算符“\*”可以引用一个变量。  
如：当ptr已经指向变量i后，\*ptr就等同于i。

When a pointer of 'ptr' is pointed to a valuable 'i', then  
'\*ptr' is equivalent to 'i'.

`*ptr = 15;`



进一步理解“&”和“\*”：“&”运算和“\*”运算是一对互逆运算。

当 `ptr=&i` 时

`&*ptr`  $\longleftrightarrow$  `&i`  $\longleftrightarrow$  `ptr`  
`*&i`  $\longleftrightarrow$  `*ptr`  $\longleftrightarrow$  `i`

等价于

## (二) 指针的赋值运算 (=) Assignment to pointer

指针的赋值运算：就是把地址赋值给指针变量。

指针的赋值运算可以是以下三种方式：

(1)使用取地址运算符，把地址值赋值给指针变量。

```
如： int i, *pi; pi=&i;
```

(2)把指针变量的值赋给另一个指针变量。

```
如： int i,*pa,*pb; pa=&i; pb=pa;
```

(3)给指针变量赋值为符号常量NULL。

```
如： int *pi; pi=NULL;
```

说明：NULL是一个空指针，表示该指针变量的值没有意义。作用是为了避免对没有被初始化的指针变量的非法引用。NULL的定义在“stdio.h”中。



## 说明:

- (1)在定义指针变量时，可以立即将一个地址值赋给指针变量，这就是指针变量的初始化。指针变量的初始化也是指针的赋值运算。

如: `float flt, *f_ptr=&flt;`

注意: 这不是给 `*f_ptr` 赋值。

`float flt,*f_ptr;`

`f_ptr=&flt;`

- (2)指针变量间的赋值和引用应保证基类型相同。

若有定义: `int *p , i ; float *q , x ;`

则: `q=&i ;` ✗ `p=&x ;` ✗

## (三) 移动指针的运算 **Pointer operation**

- 指针的加减运算（+、-） **plus and minus**
- 指针的自加自减运算（++，--，+=，-=）  
**increment and decrement**

### 1. 指针的+、-运算

+、- { 指针 ± 整数 `pointer ± integer`  
      指针 - 指针 `pointer1 - pointer2`

## 说明:

(1)指针与整型值加减的结果是指针，表示使该指针指向该指针下移或上移该整型值个数的存储单元之后的内存地址。存储单元的大小就是该指针的数据类型所需的内存大小。

例如： $\text{ptr} + n$  (指针 $\text{ptr}$ ， $n$ 为整数)这个指针值代表的内存单元的地址是： $\text{ptr} + n * d$  (其中 $d$ 是指针所指向变量的数据类型所占内存字节数)，即指针移动了 $n$ 个元素。

如果  $\text{ptr}$  指向某个数组  $a[m]$ ，则  $\text{ptr} + i$  指向数组  $a$  的第  $i$  个数组元素，即指向  $a[i]$ 。

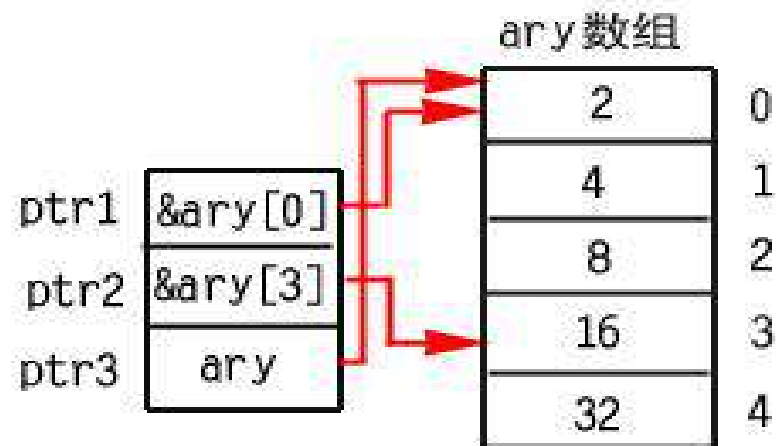
(2)指针与指针的加运算毫无意义，所以指针与指针没有加运算。同样，指针与指针也没有相乘、相除等运算。

(3)指针与指针的减运算要求相减的两个指针属于同一类型，其结果是整数，表示两个指针之间的数据的个数。其结果值的计算公式是：

$$\text{ptr1} - \text{ptr2} = (\text{ptr1 的值} - \text{ptr2 的值}) / \text{指针的数据类型所占的字节数}$$

例如：

```
int *ptr1,*ptr2,*ptr3, x;  
int ary[5]={2,4,8,16,32};  
ptr1 = &ary[0];  
ptr2 = &ary[3];  
ptr3 = ary;  
x=ptr2-ptr1;
```



x 的值是3

## 2. 指针的++、--、+=、-= 运算

++、+=：是移动指针到下一个或下几个存储单元。

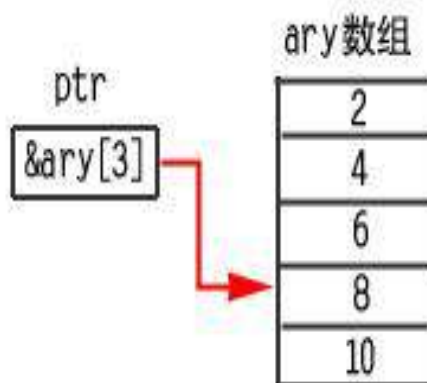
--、-=：是移动指针到上一个或上几个存储单元。

例如：`int *ptr, ary[5]={2,4,6,8,10} ;`

`ptr=ary; ptr+=3; ptr--;`



`ptr=ary;`



`ptr+=3;`



`ptr--;`

想一想：\*ptr++和(\*ptr)++有什么区别？

## (四) 指针表达式 Pointer expression

像其它变量一样，指针变量也可以用于表达式。

Like other variables, pointer variables can be used in expressions. For example, if p1 and p2 are properly declared and initialized pointers, then the following statements are valid.

`y=*p1**p2;`//equal to `(*p1)*(*p2)`

`sum=sum+*p1;`

`z=5*- *p2/ *p1;`//equal to `5*(- *p2)/(*p1)`

`*p2=*p2+10;`

The following are wrong.

`z=5*- *p2/*p1;`

`p1/p2` or `p1*p2` or `p1/3` or `p1+p2`

### 三、指针变量作为函数参数

指针可以用作函数参数，这在希望通过调用函数来改变主调函数的参数值时非常有用。

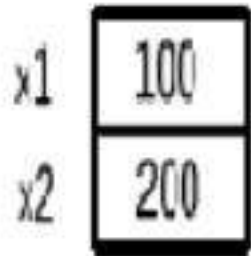
例如：用指针变量编写实现两个数的交换的函数

```
void swap(int *p1, int *p2);  
main()  
{int x1=100,x2=200;  
printf("x1=%d,x2=%d\n",x1,x2);  
swap(&x1,&x2);  
printf("x1=%d,x2=%d\n",x1,x2);  
}
```

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

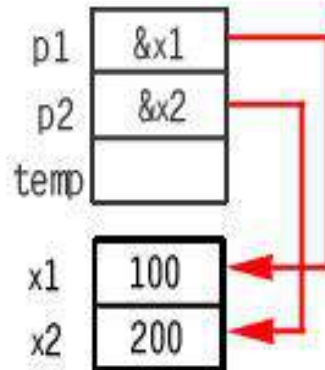
# 图示交换过程中存储单元内容的变化:

①



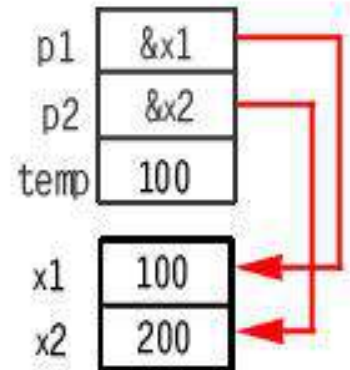
x1=100, x2=200

②



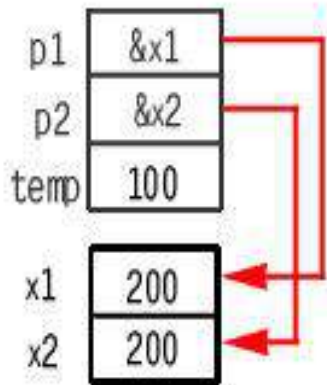
进入 swap 函数

③



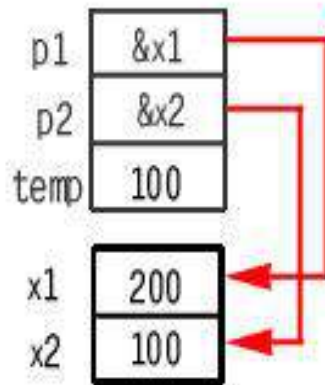
temp=\*p1;

④



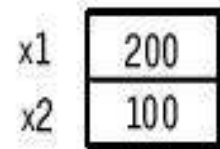
\*p1=\*p2;

⑤



\*p2=temp;

⑥



退出 swap 函数



**想一想：**如果函数的参数不用指针而用整型变量，能否实现值的交换？为什么？

通过函数调用得到n个要改变的值的方法：

- (1)在主调函数中设n个变量，将这n个变量的地址传给所调用的函数的形参指针；
- (2)通过形参指针变量，通过取值运算符“\*”改变该n个变量的值；
- (3)回到主调函数中就可以使用这些改变了值的变量。

**注意：**不能企图通过改变指针形参的值而使指针实参的值改变。

如：上例swap函数中不能写成：

```
int* temp; temp=p1; p1=p2; ;p2=temp;
```

## § 10.3 数组的指针和指向数组的指针变量

### The constant pointer of the array and the pointer variable pointing to the array

#### 一、概念

`int a[5];`

数组的指针：是数组的起始地址。  $A \leftrightarrow \&a[0]$

The constant pointer of the array is the first element (index 0) of the array.

数组元素的指针：是数组元素的地址。  $\&a[0], \&a[3]$

当指针变量指向数组或数组元素时，它就是指向数组的指针变量。

Pointer to array elements: is the address of an array element.

## C规定:

- (1)数组名代表数组的首地址（起始地址），它也是数组第一个元素的地址。
- (2)当指针变量p指向数组的某个元素时，p+1指向该元素的下一个元素。假设一个整型元素占四个字节，p+1是使p的地址加4个字节。

如： `int a[10], *p ;`

则： `p=a ;` 与 `p=&a[0];`等价

称指针变量p指向数组元素a[0]

**p+i、 a+i 、 &a[i] 都是a[i]的地址。**

## 二、数组元素的引用

### 1. 用下标法引用数组元素

如: `a[3]=45;`   `b[2][5]=200;`

### 2. 用指针法引用数组元素

假如: `int a[10], *p, i;`

`p=a;`

则: (1) `*(p+i)`、`*(a+i)`则代表元素`a[i]`

`*(a+3)=45;`  $\Leftrightarrow$  `*(p+3)=45;`

(2) `*(p+i)`也可以写成`p[i]`

(3) `*(p+i)`、`*(a+i)`、`a[i]`、`p[i]` 等价,

都代表数组`a`的第`i+1`个元素`a[i]`。

## 程序举例：输出10个元素数组中的全部元素。

方法一：下标法。 (by subscripts)

```
main()
```

```
{  int a[10]={54,65,8,2,3,56,8,21,57,98}, i;  
    for(i=0;i<10;i++) printf("%4d",a[i]);  
}
```

方法二：通过数组名计算数组元素地址，找出元素的值。 (by array name and the integer)

```
main()
```

```
{  int a[10]={54,65,8,2,3,56,8,21,57,98},i;  
    for(i=0;i<10;i++) printf("%4d",*(a+i));  
}
```

### 方法三：用指针变量指向数组元素 (by pointer)

```
main()
{   int a[10]={54,65,8,2,3,56,8,21,57,98},*p,i;
    p=a;
    for(printf("\n"),i=0;i<10;i++) printf("%4d",*p++);
}
```

以上三种方法，利用指针变量效率最高。

说明：(1)指针变量与数组名的区别：

**Difference between the pointer variable and the array name.**

指针变量是地址变量，其内容可以在程序运行过程中被改变。

数组名是地址常量，一旦被定义，它的地址就不能再改变了。

**The pointer variable is an *address variable*, and the array name is an *address constant*.**

例如： `int i, *p, a[6];`      则： `p=&i;`    ✓

~~`a=&i;`~~    ~~`a++;`~~    ~~`a+=i;`~~

不能给常量赋值

## (2)利用指针变量编程时特别要注意指针变量的当前值 Pay attention to the current value of a pointer

例如：通过指针变量输入输出a数组元素。

main()

```
{ int *p,i,a[5];
```

应插入语句 p=a;

```
    p=a;
```

```
    for(i=0;i<5;i++) scanf("%d",p++);
```

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

```
    for(i=0;i<5;i++)
```

```
        printf("%6d",*p++);
```

```
}
```

**注意：** \*p++、\*(p++)、(\*p)++、\*(++p)的含义



## 附录 D 运算符和结合性

优先级	运算符	含 义	要 求 运 算 对象的个数	结合方向
1	( )	圆括号		自左至右
	[ ]	下标运算符		
	->	指向结构体成员运算符		
	·	结构体成员运算符		
2	!	逻辑非运算符	1 (单目运算符)	自右至左
	~	按位取反运算符		
	++	自增运算符		
	--	自减运算符		
	-	负号运算符		
	(类型)	类型转换运算符		
	*	指针运算符		
	&	取地址运算符		
	sizeof	长度运算符		
3	*	乘法运算符	2 (双目运算符)	自左至右
	/	除法运算符		
	%	求余运算符		
4	+	加法运算符	2 (双目运算符)	自左至右
	-	减法运算符		
5	<<	左移运算符	2 (双目运算符)	自左至右
	>>	右移运算符		
6	< <= > >=	关系运算符	2 (双目运算符)	自左至右

**\*p++::** 指针p自增，对p取值

执行顺序：先对p取值： \*p

然后指针p自增： p++

**\*(p++)**： 指针p自增，对p取值

**(\*p)++**： 先对p取值，然后使指针p所指的变量\*p自增

**\*(++p)**： 指针p自增，对p取值

执行顺序：先使指针p自增： p++

然后对p取值： \*p

### 三、数组名作函数的参数

#### Array name as the function parameter

例如：

```
void f(int arr[],int n)
{.....
}

main()
{ int array[10];
  .....
  f(array,10);
  .....
}
```

现在解释：

实际上，能够接受并存放地址值的只能是指针变量。因此，C编译系统都是将形参数组名作为指针变量来处理的。上例中**f(int arr[ ],int n)**等价于**f(int \*arr, int n)**。

$$\text{arr}[i] \Longleftrightarrow *(\text{arr}+i)$$

使用形参数组的概念只是为了与实参数组对应，直观，便于理解而已。

例：从10个数中找出其中最大值和最小值（用数组）。

Find the maximum and minimum value from 10 numbers.

```
main()
{ void max_min(int a[],int n,int *max,int *min);
  int i,arr[]={2,4,1,6,7,32,45,75,45,90},max,min;
  printf("The original array=");
  for(i=0;i<10;i++) printf("%5d",arr[i]);
  max_min(arr,10,&max,&min);
  printf("max=%d ,min=%d",max,min);
}

void max_min(int a[], int n, int *big, int *small)
{ int i;
  *big=*small=a[0];
  for(i=1;i<n;i++)
  { if(*big < a[i]) *big = a[i];
    if(*small > a[i]) *small = a[i]; }
}
```

The diagram illustrates the equivalence between array notation and pointer notation. It features three arrows: one from `a[i]` in the `for` loop of `main()` to `*big` in the `max_min` function, another from `arr[i]` in the `printf` statement to `*small` in the `max_min` function, and a third from `a[0]` in the `max_min` function to `a[i]` in the `if` statements. To the right of the `if` statements, the text `a[i] ⇔ *(a+i);` is written in orange, indicating that `a[i]` is equivalent to `*(a+i)`.

上例中如果形参数组用指针变量，则程序如下：

```
main()
{void max _min(int *,int ,int *,int *);
  int i,arr[10]={2,4,1,6,7,32,45,75,45,90},max,min;
  for(printf("The original array="),i=0;i<10;i++)
    printf("%5d",arr[i]);

  max_min(arr,10,&max,&min);
  printf("max=%d ,min=%d",max,min);
}
```

```
void max_min(int *x, int n, int *big, int *small)
{  int i;
  * big =* small =*x;
  for(i=1; i<n; i++)
  {  if(* big <*(x+i)) * big =*(x+i);
    if(* small >*(x+i)) * small =*(x+i); }
}
```

## 数组名做函数参数小结:

如果有一个实参数组，想在函数中改变此数组的元素的值，实参与形参都可用数组名或指针变量。其对应关系有以下4种情况：

- (1)实参与形参都用数组名；
- (2)实参用数组名，形参用指针变量；
- (3)实参、形参都用指针变量；
- (4)实参为指针变量，形参用数组名。

**注意：**用指针变量作实参时一定要有确定的值。

## 例a：实参和形参都用数组名

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

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

## 例b：实参用数组名，形参用指针变量

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

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

## 例c: 实参、形参都用指针变量

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

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

## 例d: 实参为指针变量，形参用数组名

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

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



**习题10.1：输入3个整数，按由小到大的顺序输出。**

**Output 3 digits in increasing sequence.**

```
void swap(int *p, int *q);
main()
{  int a,b,c,*p1=&a,*p2=&b,*p3=&c;
   scanf("%d%d%d",p1,p2,p3);
   if(*p1>*p2) swap(p1,p2);
   if(*p1>*p3) swap(p1,p3);
   if(*p2>*p3) swap(p2,p3);
   printf("\n%6d %6d%6d",*p1,*p2,*p3);
}
void swap(int *p,int *q)
{  int x;
   x=*p;*p=*q;*q=x;
}
```

**习题10.14: 将n个数按输入时顺序的逆序排列, 用函数实现。 Arrange n digits in inverse sequence to their import.**

```
void inverse(int *a,int n);
```

```
main()
```

```
{ int i,a[10]={2,4,1,6,7,32,45,75,450,89};
```

```
  for(puts(" "), i=0;i<10;i++) printf("%5d",a[i]);
```

```
  inverse(a,10);
```

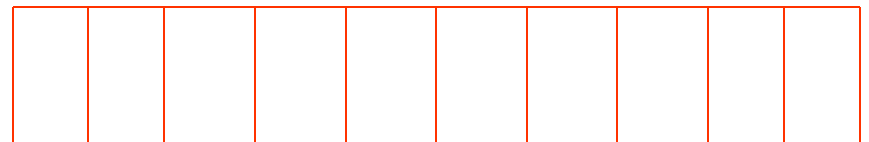
```
  for(puts(" "), i=0;i<10;i++) printf("%5d",a[i]);
```

```
}
```

```
void inverse(int *x,int n)
```

**0 1 2 3 4 5 6 7 8 9**

```
{ int *i=x,*j=x+n-1,t;
```



```
  for( ;i<j;i++,j--)
```

```
  { t=*i; *i=*j; *j=t; }
```

**\*i**

**\*j**

```
}
```

## 10.4: 有n个数，使前面各数顺序向后移m个位置。

```
main()
{int num[20],m,n,i;
printf("\nm= n="); scanf("%d%d",&m,&n);
for(i=0;i<n;i++) scanf("%d",num+i);
move(num,n,m);
for(i=0;i<n;i++) printf("%6d",*(num+i));
}

move(int *pnum,int n,int m)
{int t,i,j;
for(i=0;i<m;i++) /* m是移动轮次*/
    {t=*(pnum+n-1);
    for(j=n-1;j>0;j--) *(pnum+j)=*(pnum+j-1);
    *pnum=t; /* n是移动次数*/
    }
}
```

## 作业☺（均要求用指针和函数实现）

- (1) 任意输入4个互不相等的整数，按由大到小的顺序输出；
- (2) 在主函数中定义数组a和b，并用若干个实数将b初始化。写一个函数 `select(int n, double a[], double b[], double x)`，它将数组b中大于x的数顺序复制到数组a中，并在主函数中将a数组的内容输出。a数组的内容为空，n是两个数组的大小。（提示：a和b数组分别用两个计数器i和j计算下标，一般 $i \leq j$ ）

## 四、二维数组的指针

### 1. 二维数组的地址概念

在C语言中，一个二维数组可以看成是一个一维数组，该一维数组中的每个元素又是一个包含若干元素的一维数组。

假如有定义：`int a[3][4];`

则C语言编译程序认为a数组是由a[0],a[1],a[2]3个元素组成的一维数组，而a[0]和a[1],a[2]又分别是包含4个元素的一维数组名，分别代表a数组各行元素的起始地址(即a[0]是第0行元素的首地址，a[1]是第1行元素的首地址...)。因此，a和a[i](i=0,1,2)是两个基类型不同的指针常量。

# 二维数组的数组元素（一维数组）的首地址的运算 （各班长的地址的运算）

```
int a[3][4]
```

首地址

数组元素

二维  
数组  
一维  
数组

排

a（排长）

0班

a[0]    \*(a+0)    a[0][0]   a[0][1]   a[0][2]   a[0][3]

1班

a[1]    \*(a+1)    a[1][0]   a[1][1]   a[1][2]   a[1][3]

2班

a[2]    \*(a+2)    a[2][0]   a[2][1]   a[2][2]   a[2][3]

班长

战士

## 2. 通过地址引用二维数组元素

假如有定义: `int a[3][4], i, j; (0 ≤ i < 3, 0 ≤ j < 4)`

则 `a[i]` 和 `*(a+i)` (无条件等价) 都是第 `i` 行数组的首地址, 也是第 `i` 行第 0 列 (第 1 列) 元素的地址; 那么

第 <code>i</code> 行地址	第 <code>j</code> 列地址	<code>a[i][j]</code> 地址	<code>a[i][j]</code> 元素
----------------------	----------------------	-------------------------	-------------------------

<code>a[i]</code>	<code>[j]</code>	<code>&amp;a[i][j]</code>	<code>a[i][j]</code>
-------------------	------------------	---------------------------	----------------------

<code>a[i]</code>	<code>j</code>	<code>a[i]+j</code>	<code>*(a[i]+j)</code>
-------------------	----------------	---------------------	------------------------

<code>*(a+i)</code>	<code>j</code>	<code>*(a+i)+j</code>	<code>*(*(a+i)+j)</code>
---------------------	----------------	-----------------------	--------------------------

<code>*(a+i)</code>	<code>[j]</code>	<code>*(a+i) [j]</code>	<code>*(*(a+i))[j]</code>
---------------------	------------------	-------------------------	---------------------------

<code>&amp;a[0][0]+4*i+j</code>	<code>*(&amp;a[0][0]+4*i+j)</code>
---------------------------------	------------------------------------

# 一维数组的数组元素及地址运算

## (各班战士及地址运算)

0班**长**地址                       $a[0] \quad *(a+0)$

0班j战士的地址               $*(a+0) + j$

0班j战士  $*(*(a+0) + j) \mid a[0][0] \quad a[0][1] \quad a[0][2] \quad a[0][3]$

1班**长**地址                       $a[1] \quad *(a+1)$

1班j战士的地址               $*(a+1) + j$

1班j战士  $*(*(a+1) + j) \mid a[1][0] \quad a[1][1] \quad a[1][2] \quad a[1][3]$

2班**长**地址                       $a[2] \quad *(a+2)$

2班j战士的地址               $*(a+2) + j$

2班j战士  $*(*(a+2) + j) \mid a[2][0] \quad a[2][1] \quad a[2][2] \quad a[2][3]$



### 3. 通过一个行指针变量引用二维数组的元素

定义一个指针变量，它指向由m个元素组成的一维数组：

类型标识符 (\*指针变量名)[m];

注意：\*p两侧的圆括号不可缺少。

例如：假若有语句

```
int a[3][4], (*p)[4];  
p=a;
```

则：

- (1) p是一个指针变量，它指向由4个整型元素组成的一维数组。
- (2) p指向a数组，p+1指向数组a的下一行首地址，a和p的基类型相同，则a数组中任意元素a[i][j]还可以如下表示：

\* (p[i]+j)、\* (\* (p+i)+j)、(\* (p+i))[j]、p[i][j]

例：使用行指针变量访问九九乘法表的数组元素。

```
main()
```

```
{ float fa[9][9], (*pf)[9]=fa;
```

```
    int i,j;
```

```
    for(i=0; i<9; i++)
```

```
        for(j=0; j<=i; j++) *(* (pf+i)+j)=(i+1)*(j+1);
```

```
    for(i=0;i<9;i++)
```

j= 0 1 2 3 4

```
        for(j=0;j<=i;j++)
```

i= 0 1

```
            printf("%f", *(* (pf+i)+j));
```

1 2 4 **(i+1)\*(j+1)**

```
//或者 printf("%f", fa[i][j]);
```

2 3 6 9

```
}
```

3 4 8 12 16...

**习题10.9：写一函数，将一个 $3 \times 3$ 的矩阵转置。**

```
main()
```

```
{ void transpose(int (*p)[3]); //或者int p[][3]
  int a[3][3]={1,2,3,4,5,6,7,8,9},i,j;
  for(i=0;i<3;i++)
    for(puts(""), j=0; j<3; j++) printf("%6d",a[i][j]);
  transpose(a);
  for(i=0;i<3;i++)
    for(puts(""), j=0; j<3; j++) printf("%6d",a[i][j]);
}
transpose(int (*p)[3])//或者int p[][3]
{ int i, j, t;
  for(i=0; i<3; i++)
    for(j=0; j<i; j++)
      { t=*(*(p+i)+j);  *(*(p+i)+j)=*(*(p+j)+i);
        *(*(p+j)+i)=t;  }
}
```

## § 10.4 字符串指针和指向字符串的指针变量

字符串指针：字符串的首地址。

字符指针变量：指向字符串的指针变量。

字符指针变量定义的一般形式：

```
char *指针变量名;
```

字符指针变量定义时可以赋初值。

例如： `char *strp="Hello!";`

定义了一个指向字符串的指针变量，并将字符串的首地址赋值给strp指针变量，即：使指针变量strp指向字符串首地址。

## C程序访问字符串有以下两种方法:

### 1. 用字符数组存放一个字符串

例如: `char s []="I am a student.";`

字符串输出语句可写成:

(1) `printf("%s\n",s);`

(2) `for(i=0; s[i]!='\0'; i++) printf("%c", s[i]);`

(3) `for(i=0; s[i]!='\0'; i++) printf("%c",*(s+i));`

### 2. 用字符指针指向一个字符串

例如: `char *ps="I am a student.";`

字符串输出语句可写成:

`for(; *ps!='\0'; ps++) printf("%c", *ps);`

说明:

## (1) 字符数组与字符指针变量区别

字符数组由若干个元素组成，每个元素中放一个字符。而字符指针变量中存放的是地址（字符串的首地址），决不是将字符串放到字符指针变量中。

## (2) 赋初值的方法

字符数组和字符指针变量都可以在定义时赋初值，但以下方法对字符数组非法，对字符指针变量合法：

```
char s[10];
```

```
s="hello!"; ❌
```

```
char *ps;
```

```
ps="hello!"; ✓
```

### (3) 先赋值后使用

字符指针变量必须先赋值后使用，否则对系统构成危险。因此，不可以如下编程：

```
char *ps; scanf("%s", ps); ✗
```

而应写成：

```
char *ps,s[10]; ps=s; scanf("%s", ps); ✓
```

### (4) 字符数组名与字符指针变量的区别

字符数组名是指针常量，表示一个字符串的首地址，该地址不能改变。

而字符指针变量的值是可以改变的，它可以代表不同的字符串。

(5)若定义了一个指针变量使它指向一个字符串，就可以用下标形式引用指针变量所指字符串中的字符。

如： `char *a="I love China!"; printf("%c",a[5]);`

(6)用指针变量指向一个格式字符串，可以用它代替 `printf` 函数的格式字符串。

如： `int a; char *format; format="a=%d\n";`

则： `printf(format,a);` 相当于 `printf("a=%d\n",a);`

(7)若把字符指针指向字符数组，应注意数组的长度。

如： `char str1[10],*ps=str1; ps[10]='\0';` ❌



程序举例：将字符串a复制为字符串b。

```
char a[]="I am a Chinese.", b[20];
```

```
.....
```

```
string_copy(a,b);
```

```
puts(b);
```

```
.....
```

```
void string_copy(char *a,char *b)
```

```
{
```

```
    int i=0;
```

```
    do
```

```
    { *(b+i)=*(a+i);
```

```
      i++;
```

```
    }while(*(a+i)!='\0');
```

```
    b[i]='\0';
```

```
}
```

其中，循环语句可改为

```
for(i=0;*(a+i)!='\0';i++)
```

```
    *(b+i)=*(a+i);
```

上例程序还可写成:

```
char a[]="I am a Chinese.", b[20];
```

```
.....
```

```
string_copy(a,b);
```

```
puts(b);
```

```
.....
```

```
void string_copy(char *a,char *b)
```

```
{char *p1=a,*p2=b;
```

```
    int i=0;
```

```
    while((*p2=*p1)!='\0')
```

```
    {p2++;p1++;}
```

```
}
```

其中，循环语句可改为

```
while((*p2++=*p1++)!='\0') ;
```

## § 10.5 指针函数和函数的指针

### 一、指针函数

返回指针值的函数称作指针函数。

指针函数定义函数首部的一般形式：

类型名 \*函数名（参数表）

含意：函数的返回值是一个指针，它指向所定义类型的数据。

例如： `int *a(int x, int y); /*函数原型声明*/`

含意：a是函数名，调用它以后能得到一个指向整型数据的指针（地址）。

例：编写能返回结果串地址的串拷贝函数。

```
#include "stdio.h "
main()
{ char *strcpy1(char* str1,char*str2);
  char *ps, s1[80]="Tom is a boy";
  ps=strcpy1(s1,"Jane is a girl");
  puts(ps);
}
char *strcpy1(char *str1,char *str2)
{ char *s=str1;
  while(*str2) *str1++=*str2++;
  *str1='\0';
  return s;
}
```

## 二、函数的指针及指向函数的指针变量

### 1. 函数的指针（地址）概念

每一个函数都占用一段内存，在编译时，被分配一个入口地址，这个入口地址就称为函数的指针。

### 2. 指向函数的指针变量

定义指向函数的指针变量的一般形式：

类型标识符 (\*指针变量名)( );

例如：float (\*p)( );

含意：定义了p是指向函数的指针变量，函数的返回值是float类型。

**注意：** (1) `(*p)()`表示定义一个指向函数的指针变量，它不是固定指向哪个函数。

(2) 对指向函数的指针变量，`p+n`, `p++`, `p--`等运算是无意义的。

(3) 注意区别 `int (*p)()`、`int *p()`

### 3. 对指向函数的指针变量赋值

将一个函数的函数名（代表入口地址）赋值给指向函数的指针变量，也称该指针变量指向了这个函数。

```
如： int max(int x,int y);/*函数的原型声明*/  
      int (*p)();  
      p=max;
```

**注意：** 赋值时，只需给出函数名而不必给出参数。

## 4. 函数的调用

函数的调用 { 通过函数名调用函数  
                  通过指向函数的指针调用函数

当指向函数的指针变量p指向某一函数时  
(即指针变量p被赋值)，调用函数的形式为：

**( \*p ) ( 实参1, 实参2, ..... )**

即，只需将 **( \*p )** 代替函数名即可。

```
例如：若有定义  int max(int x,int y);  
                  int (*p)() ,a,b,c ;  
                  p=max;  
则可有语句  c= (*p) (a,b) ;
```

例：通过函数指针调用两个不同函数。

```
#include<math.h>
#define PI 3.1415926
double y(double);
double cos(double);
double table(double (*f)(), double, double, double);

void main()
{
    printf("Table of y(x)=2*x*x-x+1\n");
    table(y,0.0,2.0,0.5);
    printf("Table of cos(x)\n");
    table(cos,0.0,PI,0.5);
}
```



```
double table(double (*f)(), double min, double max,  
double step);  
{double x,value;  
  for(x=min;x<=max;x+=step){  
    value=(*f)(x);  
    printf(“%5.2f  %10.4f\n”, x, value);  
  }  
}
```

```
double y(double x)  
{  return(2*x*x-x+1);}
```

```
double cos(double x)  
{  return(cos(x));}
```

## § 10.7 指针数组和指向指针的指针

### 一、指针数组

指针数组：就是数组中每个元素是基类型相同指针变量。

定义指针数组的一般形式：

类型标识符 \*指针变量名[常量表达式];

例如： `int *p[2];`

含意： `p`是一个一维指针数组，每个元素都是一个指向整型变量的指针变量。可以将整型变量的地址赋值给元素`p[0]`或`p[1]`。

注意：区别 `int *p[2]` 、 `int (*p)[2]` 的含意。

## 二、指针数组的应用

### 1. 利用指针数组处理多个字符串

方法(1): 先用指针数组指向字符数组, 再处理。

```
main()
{
    char name[][80]={"aaa","bbb","ccc","ddd","eee"};
    char *pname[10], i ;
    for(i=0;i<5;i++) pname[i]=name[i];
    for(i=0;i<5;i++) puts(pname[i]);
}
```

```
main() /*与用行指针解决上述问题比较*/
{
    char name[][80]={"aaa","bbb","ccc","ddd","eee"};
    char (*pname)[80]=name, i ;
    for(i=0;i<5;i++) puts(pname[i]);//pname[i]=pname+i
}
```

## 方法(2): 用指针数组指向字符串常量。

```
main()
{
    char *pname[10]={"aaa","bbb","ccc","ddd","eee"};
    int i ;
    for(i=0;i<5;i++) puts(pname[i]);
}
```

## 2. 用指针数组指向动态内存

### void指针类型介绍:

ANSI新标准增加了void指针类型，它用于定义一个指针变量，但不指定它是指向哪种类型数据。在将它的值赋给另一个指针变量时要进行强制类型转换使之适合于被赋值的变量的类型。

例如: `char *p1; int a=2; void *p2;`  
`p2=&a;`  
`p1=(char *)p2;`

void指针变量定义

强制类型转换

同样, 可用 `p2=(void *)p1;` 将p1的值转换成void 指针类型。

可以将一个函数定义为void 指针类型。表示该函数返回的是一个地址, 它指向空类型, 如需要引用此地址, 也需要根据情况进行类型转换。

如, 库函数: `void *malloc(unsigned size);`  
功能: 分配size字节的存储区。

函数调用例: `char *p; p=(char *)malloc(60);`

分配60字节的存储区, 并返回该内存区的地址, 并赋值给p。

**例：用指针数组指向动态内存，进行字符串的排序。**

```
void sort(char *ps[ ],int n);  
#define N 5  
main()  
{  
    char *ps[100],i;  
    for(i=0;i<N;i++)  
        ps[i]=(char*)malloc(60);  
    for(i=0;i<N;i++)  
        gets(ps[i]);  
    sort(ps,N);  
    for(i=0;i<N;i++)  
        puts(ps[i]);  
}
```

```
void sort(char *ps[],int n)  
{    int i, k;  
    char *p, s[100];  
    for(i=0;i<n-1;i++)  
        for(k=i+1;k<n;k++)  
            if(strcmp(ps[i],ps[k])<0)  
                { p=ps[i];  
                  ps[i]=ps[k];  
                  ps[k]=p;  
                }  
}
```

### 三、指向指针的指针变量

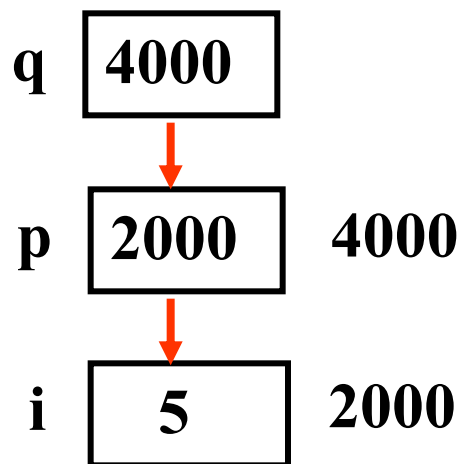
指向指针的指针变量（也称多级指针）只能存放指针变量的地址。

定义形式： 类型标识符    \*\*指针变量名；

例如： `int **q, *p, i=5;`

`p=&i; q=&p;`

含意：使指针变量p指向i，指针变量q指向p。



**注意：**要使用一个多级指针指向目标值，必须连续使用指针运算符“\*”。

例如：上例中\*\*q就是i的值。

## § 10.8 指针的数据类型小结

定 义	含 义	名 称
<code>int *p ;</code>	p为指向整型数据的指针变量	指针变量
<code>int *p[n] ;</code>	定义指针数组p，它有n个指向整型数据的指针元素	指针数组
<code>int (*p)[n] ;</code>	p为指向含n个元素一维数组的行指针变量	行指针变量
<code>int *p() ;</code>	p()为返回值为指针的函数（指针函数），该指针指向整型数据	指针函数
<code>int (*p)() ;</code>	p为指向函数的指针，该函数返回一个整型值	指向函数的指针
<code>int **p ;</code>	p是一个指针变量，它指向一个指向整型数据的指针变量	指向指针的指针



**习题10.2：输入3个字符串，按由小到大的顺序输出。**

```
void swap(char *ps1,char *ps2);
main()
{  char s1 []="xyz",s2 []="rst",s3 []="abc";
   if(strcmp(s1,s2)>0) swap(s1,s2);
   if(strcmp(s1,s3)>0) swap(s1,s3);
   if(strcmp(s2,s3)>0) swap(s2,s3);
   printf("\n%10s %10s %10s",s1,s2,s3);
}
void swap(char *ps1,char *ps2)
{char ps[100];
strcpy(ps,ps1);strcpy(ps1,ps2);strcpy(ps2,ps);
}
//char *ps;
//ps=ps1;ps1=ps2;ps2=ps;
```

## strcmp(字符串1, 字符串2)

- 字符串比较的规则：
  - 对两个字符串自左至右逐个字符按ASCII码值大小比较，直到出现不同的字符或遇到'\0'为止；
  - 若全部字符相同，则认为相等；
  - 若出现不同的字符，则以第一个不相同的字符的比较结果为准。
- 比较的结果由函数名带回：
  - 1) 如果字符串1=字符串2，函数值为0；
  - 2) 如果字符串1>字符串2，函数值为一正整数；
  - 3) 如果字符串1<字符串2，函数值为为一负整数；

**习题10.17: 写一函数，实现两个字符串的比较。**

**如果s1>s2输出正值，如果s1<s2输出负值。**

```
main()
{int cmps(char *,char *);
  char s1[128],s2[128];
  gets(s1);
  gets(s2);
  printf("cmps(s1,s2)=%d",cmps(s1,s2));
}
int cmps(char *s1,char *s2)
{ while(*s1&&*s2&&*s1++==*s2++) ;
  return *s1-*s2;
}
```

**同时满足下列3  
个条件时，循  
环继续**

**\*s1!='\0'**

**\*s2!='\0'**

**\*s1==\*s2**

**习题10.6：** 写一个函数，求字符串的长度，在main函数中输入字符串，并输出其长度。

```
int len(char *ps);  
void main()  
{   char str[30];  
    gets(str);  
    printf("\n len=%d",len(str));  
}  
int len(char *ps)  
{   int n=0;  
    while(*ps++) n++;  
    return n;  
}
```

**习题10.7:** 有一字符串包含n个字符，写一个函数，将字符串中从第m个字符开始的全部字符复制成为另一个字符串。

```
void main()
```

```
{  char sn[100]="fasjfkjsdfsd fjsdjkfd fjs sdf",sm[100];  
    int n=strlen(sn),m=n/2;//将数组的后半段复制  
    scopy(sn,sm,m);  
    printf("\n sn=%s \n m=%d \n sm=%s",sn,sm,m);  
}
```

```
scopy(char *psn,char *psm,int m)
```

```
{  while(*(psn+m-1)) *psm++=*(m-1+psn++);  
    *psm='\0';  
}
```

**习题10.8：** 输入一行文字，找出其中大写字母、小写字母、空格、数字以及其他字符各有多少？

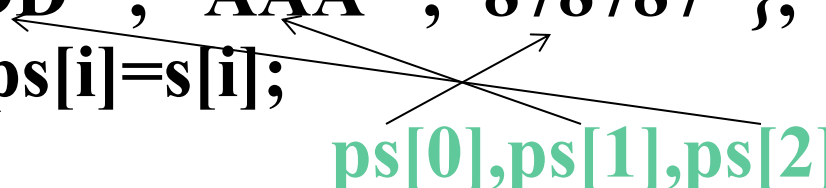
```
void main()
{ char s[100],*p;
  int lowc=0, upc=0, space=0, num=0, other=0;
  gets(s);
  p=s;
  while(*p)
  { if(*p>='A' && *p<='Z') upc++;
    else if(*p>='a' && *p<='z') lowc++;
    else if(*p>='0' && *p<='9') num++;
    else if(*p==' ') space++;
    else other++;
    p++;
  }
  printf("\n upc=%d lowc=%d num=%d space=%d other=%d",
        , upc, lowc, num, space, other);
}
```

## 习题10.11: 对10个等长的字符串排序并输出。

```
void sort(char (*p)[60]);
main()
{
    char s[10][60],i;
    for(i=0; i<10; i++) gets(s+i);
    sort(s);
    for(i=0; i<10; i++) puts(s+i);
}
void sort(char (*p)[60])
{
    int i,j;
    char s[60];
    for(i=0; i<9; i++)
        for(j=i+1; j<10; j++)//冒泡法
            if(strcmp(p[i],p[j])>0)
                { strcpy(s,p[i]); strcpy(p[i],p[j]); strcpy(p[j],s);}
}
```

## 10.12: 用指针数组处理上一个题目，字符串不等长。

```
void sort(char *p[ ],int n);
main()
{
    char *s[10]={ " DDD ", " AAA ", "878787"},*ps[10],i;
    for(i=0;i<10;i++) ps[i]=s[i];
    sort(ps,3);//
    for(i=0;i<3;i++) puts(ps[i]);
}
void sort(char *pt[ ],int n)
{
    int i, j;
    char *s;
    for(i=0;i<n-1;i++)
        for(j=i+1;j<n;j++)
            if(strcmp(pt[i],pt[j])>0)
                s=pt[i], pt[i]=pt[j], pt[j]=s;
}
```



The diagram illustrates the state of the pointer array `ps` after the first loop in `main()`. It shows three arrows originating from the string literals in the initialization of `s` and pointing to the corresponding elements in `ps`:

- An arrow from `" DDD "` points to `ps[0]`.
- An arrow from `" AAA "` points to `ps[1]`.
- An arrow from `"878787"` points to `ps[2]`.

The text `ps[0],ps[1],ps[2]` is written in green below the arrows, indicating the pointers stored in the array.



## 习题10.16: 输入一个字符串, 将其中的数字串转换成数值送数组a。

```
void main()
{  char s[128],*ps=s;
   int a[100], i, n;
   gets(s);//”abc987def6543ghijk210”
   for(i=0,n=0;*ps;ps++)//如果*ps='\0'=0, 循环终止
       if(*ps>='0' &&*ps<='9')
           n=n*10+*ps- '0' ;
       else//遇到非数字时, 数字转换完成
           {  if(n>0) a[i++]=n;//将数值存入数组a
              n=0;//重新将n置零
            }
   if(n>0) a[i++]=n;//将字符串末尾的数值存入数组
   for(n=0; n<i; n++) printf("%6d",a[n]);
}
```

## 习题10.18: 根据输入的月份输出月的英文名。

```
void main()
{  char *month[]={"January","February","March",
    "April", "May","June","July", "August",
    "Septermber", "October","Noverber",
    "Decermber"};

    int mon;
    scanf("%d",&mon);
    if(mon<1 ||mon>12) printf("month error!");
    else puts(month[mon-1]);
}
```

# 作业

(上机运行以下程序，要求使用指针)

- (1) 写一函数，将一个 $3 \times 4$ 的矩阵转置，将数据存储到一个 $4 \times 3$ 的矩阵后，将两个矩阵输出。
- (2) 输入5个字符串，按由大到小的顺序输出。
- (3) 对 $n$ 个等长的字符串排序并输出， $n$ 在2~10之间可变。
- (4) 写一个函数，接受一个已排序的整形数数组和一个整数，将该整数插入数组的适当位置，将该数组在变化前后分别输出。

1. State the following statements are true or false.  
判断下列说法是真还是假

a) Pointer variables are declared using the address operator.  
指针变量用地址运算符(&)来定义。

b) The underlying type of a pointer variable is void.  
指针变量名不能使用下划线。

c) Pointers to pointers is a term used to describe pointers  
whose contents are the address of another pointer.

指向指针的指针是这样一种参数，它用于描述一种把其他指针的地址作为内容的指针。

d) It is possible to cast a pointer to float as a pointer to integer.  
可以把float 型的指针用作指向integer型数的指针。

e) An integer can be added to a pointer.

指针可以与一个整数相加。

f) A pointer can never be subtracted from another pointer.

指针不能与指针相减。

g) When an array is passed as an argument to a function, a pointer is passed.

当把数组作为变量传递给一个函数时，就等于传递了一个指针。

h) Pointers cannot be used as formal parameters in headers to function definition.

进行函数定义时，函数头内的形参不能使用指针。

2. Fill in the blanks in the following statements.

a) A pointer variable contains as its value the \_\_\_\_\_ of another variable.

指针变量把其它变量的\_\_\_\_作为它自己的值。

b) The \_\_\_\_\_ operator is used with a pointer to dereference the address contained in the pointer.

- \_\_\_\_\_运算符用于对一个指针进行操作，对指针中保存的地址进行取值运算。

c) The \_\_\_\_\_ operator returns the value of the variable to which its operand points.

\_\_\_\_\_运算符所返回的值是变量的操作数（变量的值）所指向的值。

3. If m and n have been declared as integers and p1 and p2 as pointers to integers, then state errors, if any, in the following statements.

a) p1=&m;

b) p2=n;

c) \*p1=&n;

d) p2=&\* &m

e) m=p2-p1;

f) p1=&p2;

g) m=\*p1+\*p2++;

4. Find the error, if any, in each of the following statements.

a) int x=10;

b) int \*y=10;

c) int a, \*b=&a;

d) int m;

int \*\*x=&m;

5. Given the following declarations:

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

```
int *p1=&x, *p2=&y;
```

What is the value of each of the following expressions?

a) (\*p1)++

b) --(\*p2)

c) \*p1+(\*p2)--

d) ++(\*p2)-\*p1



## 答案

1.

a)False      b)False      c)True      d)False  
e)True      f)False      g)True      h)False

2.

a)地址      b)\*      c)\*

3.

a)True      b)数值不能赋给指针变量,  
c)不能对指针变量所指向的变量赋地址  
d)True      e)True  
f)不能将指针的地址赋值给指针      g)True

4.

a) True      b) 不能对指针变量赋普通值

c) True

d) 不能对指向指针的指针赋变量的地址

5.

a) 10

b) 9

c) 20

d) 1

## 书面习题之答案

- |                                    |             |       |             |       |
|------------------------------------|-------------|-------|-------------|-------|
| 1. B                               | 2. B        | 3. A  | 4. B        | 5. D  |
| 6. B                               | 7. C        | 8. D  | 9. 66676565 |       |
| 10. C                              | 11. D       | 12. C | 13. C       | 15. C |
| 14. (1) $t[k] \neq '\backslash 0'$ | (2) $k > 0$ |       |             |       |

# 第十三章 文 件

## File management

- 掌握文件以及缓冲文件系统、文件指针的概念；
- 学会使用文件打开、关闭、读、写等文件操作函数。

# Why we have to use file?

- 1. It becomes cumbersome(麻烦) and time consuming to handle large volumes of data through terminals.
- 2. The entire data is lost when either the program is terminated or the computer is turned off.

## § 13.1 C文件概述File in C

### 一、文件的概念 Concept of the file

文件：是指存储在外部介质上数据的集合。

**A file is a place on the disk where a group of related data is stored.**

操作系统是以文件为单位对数据进行管理的。每一个与主机相连的输入或输出设备都看作是一个文件。

### 二、文件的存在形式及分类

文件的存在形式：文件名+文件内容。

文件内容：是一个字符(字节)序列，即由一个个字符(字节)的数据顺序组成。对文件的存取是以字符(字节)为单位的，这类文件称为流式文件。

## 文件的分类: File category

根据数据在内存的组织形式不同可分为两类文件:

(1)ASCII 码文件(文本文件): 文件的每一个字节放一个ASCII代码, 代表一个字符。

### **Text File**

(2)二进制文件: 把内存中的数据按其在内存中的存储形式原样输出到文件上。

### **Binary File**

根据数据读写方式不同可分为两类文件:

{ 顺序存取文件  
{ 随机存取文件

### 三、文件的基本操作

- (1) 打开文件: 把文件名等目录信息从磁盘上读入内存并存入结构体。
- (2) 关闭文件: 把内存结构体中的文件名等目录信息写入磁盘。
- (3) 写文件: 向文件写内容。
- (4) 读文件: 从文件读内容。

### 四、文件的输入输出方式

C语言对文件进行处理分为:

{ 非缓冲文件系统 no buffer file system  
{ 缓冲文件系统 buffer file system

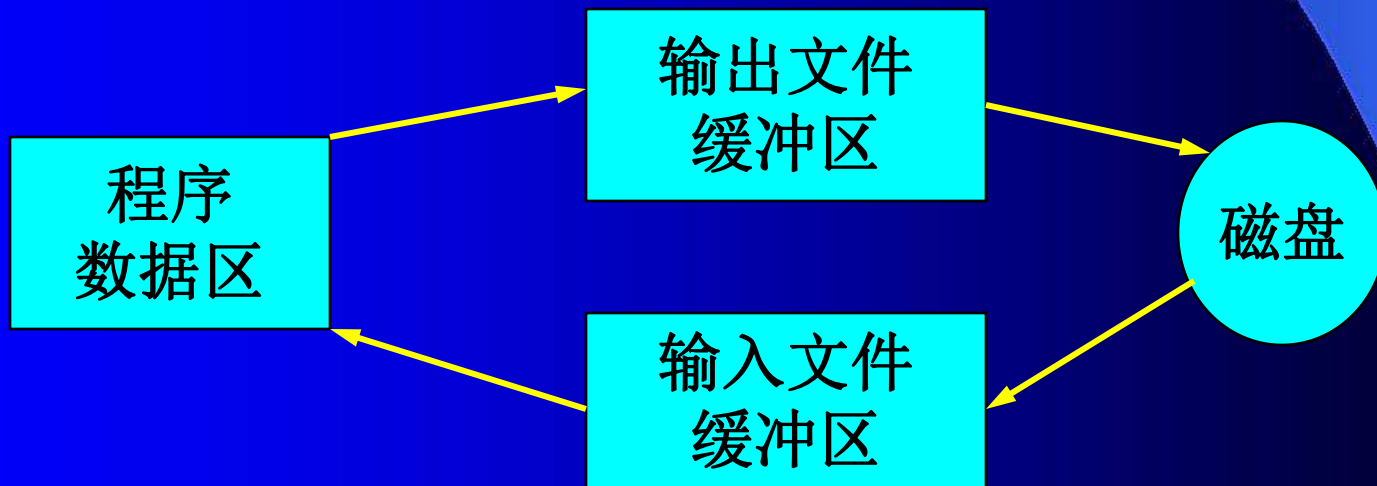


(1)非缓冲文件系统：程序与外设直接进行数据交换。

特点：每读写一次都要启动外设。

(2)缓冲文件系统：程序与外设通过内存缓冲区进行数据交换。（常用的方式）

特点：只有内存缓冲区满(写盘)或空(读盘)才启动外设。



## § 13.2 C文件类型指针

### 一、文件类型 (**FILE type**)

每个被使用的文件都在内存中开辟一个区，用来存放文件的有关信息（如文件名，状态，当前位置等）。这些信息保存在一个结构体类型的变量中。结构体类型由系统定义为“**FILE**”。

Data structure of a file is defined as **FILE** in the library of standard I/O function definitions.

Therefore, all files should be declared as **FILE** before they are used. **FILE** is a defined data type.

## 二、定义FILE类型指针变量 FILE definition

FILE 类型的指针变量定义方法：

```
FILE *指针变量名;
```

```
FILE *pointer_variable_name;
```

例如：FILE \*fp;

含义：fp是一个指向FILE类型结构体的指针变量。

fp as a pointer to the data type FILE.

可以使fp指向某个文件的结构体变量，从而通过该结构体变量中的文件信息能够访问该文件。

## § 13.3 文件的打开和关闭函数

### 一、文件的打开函数（fopen函数）

#### (1) 调用方式:

用户定义的文件类型指针变量

```
FILE *fp;  
fp= fopen(文件名, 文件的使用方式);
```

所要打开的文件名。  
必须是用双引号引起的字符串常量。

具有特定含意的符号  
也必须由双引号引起

```
fp= fopen("filename", "mode");
```

(2)功能: 以指定的方式打开指定的文件, 若操作成功, 则返回一个指向该文件的指针, 若打开文件时出现错误, 则返回空指针NULL。

```
例:  FILE *fp;  
      fp=fopen("mydata.txt","r");
```

(2)Operation:

- Creates a new file for use.
- Opens an existing file for use.

If operating success, the function will return a pointer to the file, else it will return an empty pointer of NULL.

### (3)文件使用方式

- 基本方式字符（单独使用是对文本文件）：

**r** 只读文件(读), 如无文件, 产生出错信息。

If the purpose is 'reading', and if it exists, then the file is opened with the current contents safe otherwise an error occurs.

**w** 建立新文件(写), 如有同名文件, 删除之。

When the mode is 'writing', a file with the specified name is created if the file does not exist. The contents are deleted, if the file already exists.

**a** 写旧文件(添加), 如无旧文件, 则生成该文件。

When the mode is 'appending', the file is opened in the current contents safe. A file with the specified name is created if the file does not exist.

- 按二进制格式读写文件（基本方式字符+“b”）:

**b** 辅助说明，按二进制格式读写文件。

例如:

```
fp=fopen("mydata.txt","rb");
```

- 同时进行读写文件（基本方式字符+[“b”]+“+”）:

**+** 辅助说明，可同时进行读和写文件。

**r+** The existing file is opened to the beginning for both reading and writing.

**w+** Same as w except both for reading and writing.

**a+** Same as a except both for reading and writing.

例如: `fp=fopen("mydata.txt","r+");`

例: #include <stdio.h>

main()

{ FILE \*fp;

if((fp=fopen("mydata.txt","r")) == NULL)

{ printf("file not found.");

exit(0);

}

else

{ printf("file opened ok!");

fclose(fp); }

}

常使用此方法打  
开一个文件

exit函数的功能是关闭所有文件，终止程序运行。  
exit(0)为正常终止程序

## Instruction:

fp=fopen("mydata.txt","r")

if(fp == NULL)



程序运行结果:

如果磁盘上存在文件mydata.txt, 屏幕上会显示  
**file open ok!**, 否则屏幕上显示**file not found.**

## 二、文件的关闭函数 (fclose函数) CLOSE A FILE

(1) 调用方式:

```
fclose(文件指针);  
fclose(file_pointer)
```

(2) 功能：关闭由文件指针指定的文件，防止它被误用；同时把缓冲区中的数据（未装满缓冲区的数据）输出到磁盘上，释放文件指针。

A file must be closed as soon as all operations on it have been completed. This ensures that all outstanding information associated with the file is flushed out(冲洗) from the buffers and all links to the file are broken.

```
例如： FILE *fp;  
        fp=fopen("output.dat","r+");  
        ..... /* 进行读写操作 */  
        fclose(fp);
```

**注意：** fopen函数和fclose函数总是成对出现的。  
无fclose函数时会导致部分数据丢失！

## § 13.4 文件的读写函数

### Input/Output Operations on Files

#### 一、fputc函数 和fgetc函数

这两个函数是以字符为单位进行文件读写的函数。

These are analogous to **getchar** and **putchar** functions and handle one character at a time.

# 1. fputc函数

## 一般调用形式:

fputc(字符常量或变量, 文件指针);

fputc(character constant or variable, file\_pointer)

功能：把指定字符存入文件指针所指的文件中。

如果输出失败，则返回一个EOF（-1）。

（EOF是在stdio.h文件中定义的符号常量，值为-1）

例如:fputc(ch,fp); (其中ch为字符变量,fp为文件指针)

含义：将ch的值输出到fp所指向的文件中去。

Write the character contained in the character variable **ch** to the file associated with FILE pointer **fp**.

If the output failed, the fputc will return a marker of EOF.

## 2. fgetc函数

一般调用形式:

```
字符变量=fgetc(文件指针);  
  
ch=fgetc(fp);
```

功能: 从文件指针所指文件中读一个字符赋给指定的字符变量。当遇文件结束符, 返回一个文件结束标志EOF (-1)。

The file pointer moves by one character position for every operation of fgetc or fputc. The fgetc will return an end-of-file marker EOF, when end of the file has been reached.

例如: **ch=fgetc(fp);**

其中**ch**为字符变量, **fp**为文件指针。

含义:从**fp**所指向的文件中读入1个字符赋给字符变量**ch**。

例：阅读程序。

```
#include "stdio.h"
```

```
#define NULL 0
```

```
main()
```

```
{ FILE *fp;
```

```
int count=0; char c;
```

```
if((fp=fopen("file.txt","r")) ==NULL)
```

```
{ printf("Can't Open File \n"); exit(0); }
```

```
do {c=fgetc(fp); count++;} while(c!=EOF);
```

```
count--;
```

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

```
fclose(fp);
```

```
}
```

这是读一个文本文件中字符，并判断文件是否结束的常用方法

若文件file.txt的内容是：abcd1234ABCD

输出结果： 12

# 关于文件读写状态的检测

## Error Handling During I/O Operations

函数调用是否成功，可有两种手段来检测：

1. 由函数的返回值来确定
2. 用文件状态检测函数feof、ferror、fclearerr

**feof(fp)** :测试fp所指向的文件的位置是否是文件尾。若已到达文件尾，函数返回非零值，若没有到达文件尾，则返回0。

feof(fp) will return a non-zero value after the pointer **fp** pointes to the end of file. Otherwise it will return 0.

常用程序段：

```
while(!feof(fp))  
{.....}
```

```
if(feof(fp))  
    printf("End of data.\n");
```

**feof(fp)** :测试对fp所指向的文件的操作是否出错。若出错，函数返回非零值，否则返回0。

```
if(ferror(fp)!=0)
    printf("An error has occurred.\n");
```

**fclose(fp)** :使ferror和feof函数值置0。



## 二、 fread函数和fwrite函数

一般调用形式:

```
fread(buffer,size,count,fp);
```

```
fwrite(buffer,size,count,fp);
```

其中:

**buffer**:是一个指针(起始地址)。

对fread来说,它是读入数据的存放地址。

对fwrite来说,它是输出数据的地址。

**size**:要读写的字节数。

**count**:要进行读写多少个size字节的数据项。

功能: 对fp所指的文件读写buffer所指的size\*count个字节数据。

例如: fread(f,4,2,fp); (其中f是实型数组名)

**例1 把数组a写入文件fa.txt;再从fa读入数组b。**

```
#include "stdio.h"  
main()  
{ FILE* fp;  
  int a[100]={1,5,6,78,21,34,67,87,23},b[100], i ;  
  fp=fopen("fa.txt","wb");  
  fwrite(a,sizeof(a),1,fp);  
  /* fwrite(a,sizeof(int),100,fp);*/  
  fclose(fp);  
  fp=fopen("fa.txt","rb");  
  fread(b,sizeof(a),1,fp);  
  for(puts(""),i=0;i<10;i++) printf("%6d",b[i]);  
  fclose(fp);  
}
```

### 三、 **fprintf** 函数和**fscanf**函数

这两个函数是进行格式读写文件的函数。

一般调用形式：

**fprintf**(文件指针， 格式字符串， 输出表列) ；

**fscanf**(文件指针， 格式字符串， 输入地址) ；

**fprintf**(*fp*, "control string", list) ；

*fp* is a file pointer associated with a file that has been opened for writing.

The *control string* contains output specifications for the items in the list.

The *list* may include variables, constants and strings.

功能：按指定格式将数据写到指定文件中。  
或从指定文件按格式输入数据。

The functions **fprintf** and **fscanf** perform I/O operations, except of course that they work on files.

例如： `fprintf(fp, "%d,%6.2f",i,x);`

又如： `fscanf(fp, "%d,%f",&i,&x);`

```
#include "stdio.h"

void main()
{char infile[15], outfile[15], radar[10], func[6];
FILE *fp1, *fp2;
    printf("请输入雷达数据文件名(*.txt)\n");
    scanf("%s", infile);
    if((fp1=fopen(infile, "r"))==NULL)
    {
        printf("cannot open this file\n");
        exit(0);
    }
```

```

fscanf(fp1,"%s",radar);
fscanf(fp1,"%s",func);
fscanf(fp1,"%f",&frequ);
fscanf(fp1,"%f",&rmax;
fscanf(fp1,"%f",&sigma0);
fscanf(fp1,"%f",&pd0);
fscanf(fp1,"%e",&p_fa);
fscanf(fp1,"%f",&prf);
fscanf(fp1,"%f",&tao);
fscanf(fp1,"%f",&rpm);
fscanf(fp1,"%d",&type);
fscanf(fp1,"%f",&phi_b);
fscanf(fp1,"%f",&theta_b);
fscanf(fp1,"%f",&Emax);
fclose(fp1);
.....

```

AN\_TPS-43E.txt

4000	脉冲功率(kw)
6310	最大增益
0.3125	带宽(MHz)
3	中心频率(GHz)
0.1	天线高度(km)
440	最大作用距离(km)
5	对应散射截面(m <sup>2</sup> )
0.5	对应发现概率
1e-006	对应虚警概率
1.1	水平波束宽度(deg)
4.8	垂直波束宽度(deg)
20	仰角扫描范围(deg)
6	天线转速(rpm)
1	斯威林模型(1~4)
6.5	脉冲宽度(us)
250	脉冲重复频率(Hz)
2	脉冲积累数
1.20472e+023	系统特征常数(m <sup>2</sup> )

```
printf("请输入输出数据文件名(*.txt)\n");  
scanf("%s",outfile);  
if((fp2=fopen(outfile,"w"))==NULL)  
{  
    printf("cannot open this file\n");  
    exit(0);  
}
```

```
fprintf(fp2,"Radar:%s, Function=%s, Frequency=%5.1f MHz "
        ,radar,func,frequ);
fprintf(fp2, " PRF=%5.1f Hz, Tao=%4.1f us, RPM=%4.1f r/min\n"
        ,prf,tao,rpm);
fprintf(fp2,"swerling_case:%d, rmax=%6.1f km, sigma0=%5.1fdB,
        PD0=%5.1f, P_fa=%e, N number=%d\n"
        ,swerling_case,rmax,sigma0,pd0,p_fa,N);
for(j=0;j<X;j++) fprintf(fp2,"%4.0f,dmsm ",sigma[j]);
fprintf(fp2,"\n");
for(j=0;j<Z;j++)
{
    for(i=0;i<X;i++) fprintf(fp2,"\t%6.4f",pdd[i][j]);
    fprintf(fp2,"\n");
}
fclose(fp2);
.....
}
```



## 四、 fputs函数 和fgets函数

这两个函数是以字符串为单位进行文件读写的函数。

### 1. fputs函数

一般调用形式: `fputs(字符串, 文件指针);`

功能: 向指定文件输出一个字符串。

例如: `fputs(str,fp);` (其中str是字符数组名)

含义: 向fp 所指向的文件中输出str中的字符串。

### 2. fgets函数

调用形式: `fgets(字符数组, 字符串长度, 文件指针);`

功能: 从指定文件中读入一个字符串。

例如: `fgets(str,n,fp);` (其中str是字符数组名)

含义: 从fp指向的文件读取长度为n-1的字符串, 最后加一个 ‘\0’存入字符数组str中。

## 五、文件的定位函数

### 1. rewind函数

一般调用形式: `rewind(文件指针);`

功能: 使位置指针重新返回文件的开头。

### 2. fseek函数

一般调用形式:

`fseek(文件指针, 位移量, 起始点);`

起始点: 用0、1、2或其对应的名字SEEK\_SET、SEEK\_CUR、SEEK\_END代替, 分别表示文件开始、文件当前位置、文件末尾。

位移量: 指以起始点为基点, 向前(-)或向后移动的字节数。ANSI C标准规定在数字末尾加一个字母L, 表示long型。

**功能：**把文件的读写位置指针移到指定的位置。

**例如：** `fseek(fp,128L,SEEK_SET);`

将位置指针移到离文件头128个字节处。

**例如：** `fseek(fp,-10L,2);`

将位置指针从文件末尾处向前退10个字节。

**注意：** `fseek`函数一般用于二进制文件。

### 3. `ftell`函数

一般调用形式：`ftell(文件指针);`

**功能：**返回文件位置指针的当前值。

**例如：** `pos=ftell(fp);`

获取`fp`指向文件的当前读写位置，并将其值赋给变量`pos`。

# 作业:

1. 设文件vehicle\_1.txt和vehicle\_2.txt存放了两种飞行器的雷达散射截面（RCS）数据（量纲为dBsm）。请编写程序对两组数据分别计算其算术平均值，并根据该算术平均值评价两种飞行器隐身性能的好坏。