

```
#include <stdio.h>

#define IN 1 /* inside a word */
#define OUT 0 /* outside a word */

/* count lines, words, and characters in input */
main()
{
    int c, nl, nw, nc, state;

    state = OUT;
    nl = nw = nc = 0;
    while ((c = getchar()) != EOF) {
        ++nc;
        if (c == '\n')
            ++nl;
        if (c == ' ' || c == '\t')
            state = OUT;
        else if (state == OUT) {
            state = IN;
            ++nw;
        }
    }
    printf("%d %d %d\n", nl, nw, nc);
}

#include <stdio.h>

#define IN 1 /* inside a word */
#define OUT 0 /* outside a word */

/* count lines, words, and characters in input */
main()
{
    int c, nl, nw, nc, state;

    state = OUT;
    nl = nw = nc = 0;
    while ((c = getchar()) != EOF) {
        ++nc;
        if (c == '\n')
            ++nl;
        if (c == ' ' || c == '\n' || c == '\t')
            state = OUT;
        else if (state == OUT) {
            state = IN;
            ++nw;
        }
    }
    printf("%d %d %d\n", nl, nw, nc);
}
```

课件下载：

https://github.com/xiaodongli1986/teaching_c

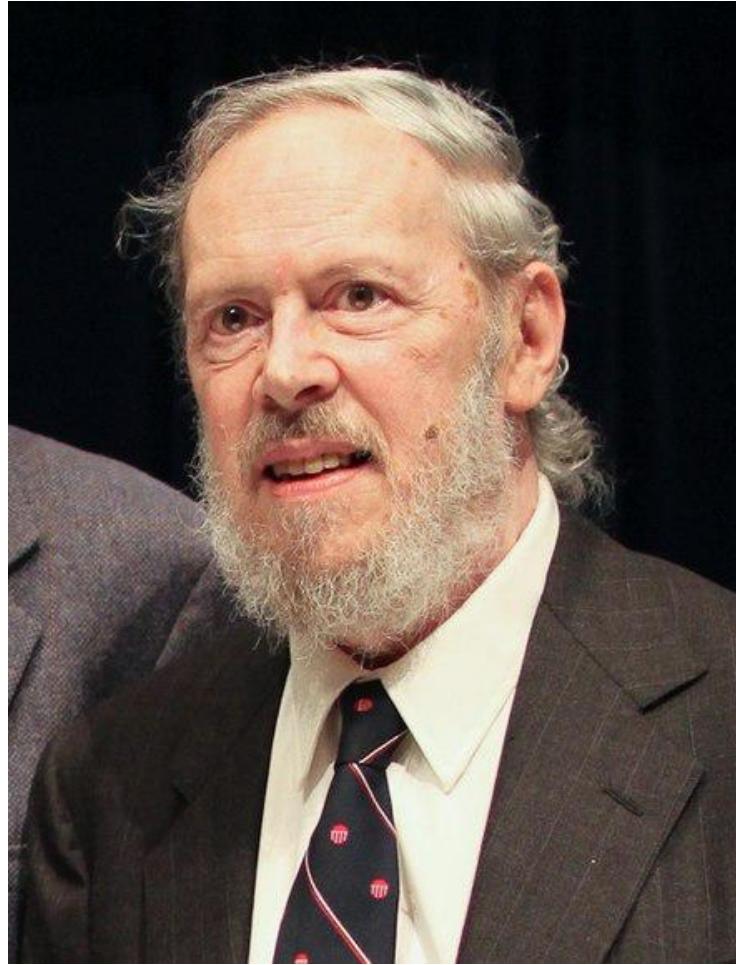
第一章

C语言概述

§1-1 C语言出现的历史背景

- C语言是国际上广泛流行的高级语言。
- C语言是在B语言的基础上发展起来的。
 - B (BCPL) 语言是1970年由美国贝尔实验室设计的，并用于编写了第一个UNIX操作系统。优点：精练,接近硬件，缺点：过于简单,数据无类型。
- 1973年贝尔实验室的D.M.Ritchie 在B语言的基础上设计出了C语言，对B取长补短，并使用C重写了Unix 第5版。

§1-1 C语言出现的历史背景



Dennis MacAlistair Ritchie
(September 9, 1941 – c. October
12, 2011) ,
美国计算科学家。

他创造了 C 编程语言；与长期合作者
Ken Thompson 一起创造了
Unix 操作系统。

§1-1 C语言出现的历史背景

- 1978年影响深远的名著《The C Programming Language》。之后，C语言先后移植到大、中、小、微型计算机上,成为最广泛的几种计算机语言之一。
- 1983年,美国国家标准化协会(ANSI)根据C语言各种版本,制定了标准ANSI C。目前流行的C语言编译系统大多是以ANSI C为基础进行开发的。
- 不同版本的C编译系统语言功能和语法规则略有差别（可以参阅有关手册）。本课程以ANSI C 为基础。

§1-2 C语言的特点

- (1) 语言简洁、紧凑,使用方便、灵活。32个关键字、9种控制语句,程序形式自由
- (2) 运算符丰富。34种运算符。数据类型丰富,具有现代语言的各种数据结构。
- (3) 语法限制不太严格,程序设计自由度大。
- (4) 允许直接访问物理地址,能进行位操作,能实现汇编语言的大部分功能,直接对硬件进行操作。兼有高级和低级语言的特点。
- (7) 目标代码质量高,程序执行效率高。只比汇编程序生成的目标代码效率低10%-20%。
- (8) 程序可移植性好(与汇编语言比)。基本上不做修改就能用于各种计算机和操作系统。

说明： 本程序的作用是输出一行信息：

This is a C program.

```
void main( )  
{  
    printf ("This is a C  
program.\n");  
}
```

/*文件包含*/
/*主函数 */
/*函数体开始
*/
/*输出语句*/

结束

说明： main- 主函数名， void- 函数类型

- 每个C程序必须有一个主函数main
- { }是函数开始和结束的标志, 不可省
- 每个C语句以分号结束
- 使用标准库函数时应在程序开头一行写：

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

说明： 输出一行信息：sum is 579

例1.2 求两数之和

```
#include <stdio.h>
void main( )      /*求两数之和*/
{
    int a,b,sum;    /*声明，定义变量为整型*/
    /*以下3行为C语句 */
    a=123; b=456;
    sum=a+b;
    printf(" sum is %d\n" ,sum);
}
```

说明： /*.....*/ 表示注释。注释只是给人看的，对编译和运行不起作用。所以可以用汉字或英文字符表示，可以出现在一行中的最右侧，也可以单独成行。 /*.....*/ 可以注释半行、一行、多行；//可以注释半行、一行。

§1-3 简单的C语言程序介绍

几点说明：

(1) 一个函数由两部分组成：

函数的首部：如主函数 void main()

函数体：花括号内的部分。

函数体包括两部分：

声明部分：int a,b,c; 可缺省

执行部分：若干个语句组成。可缺省

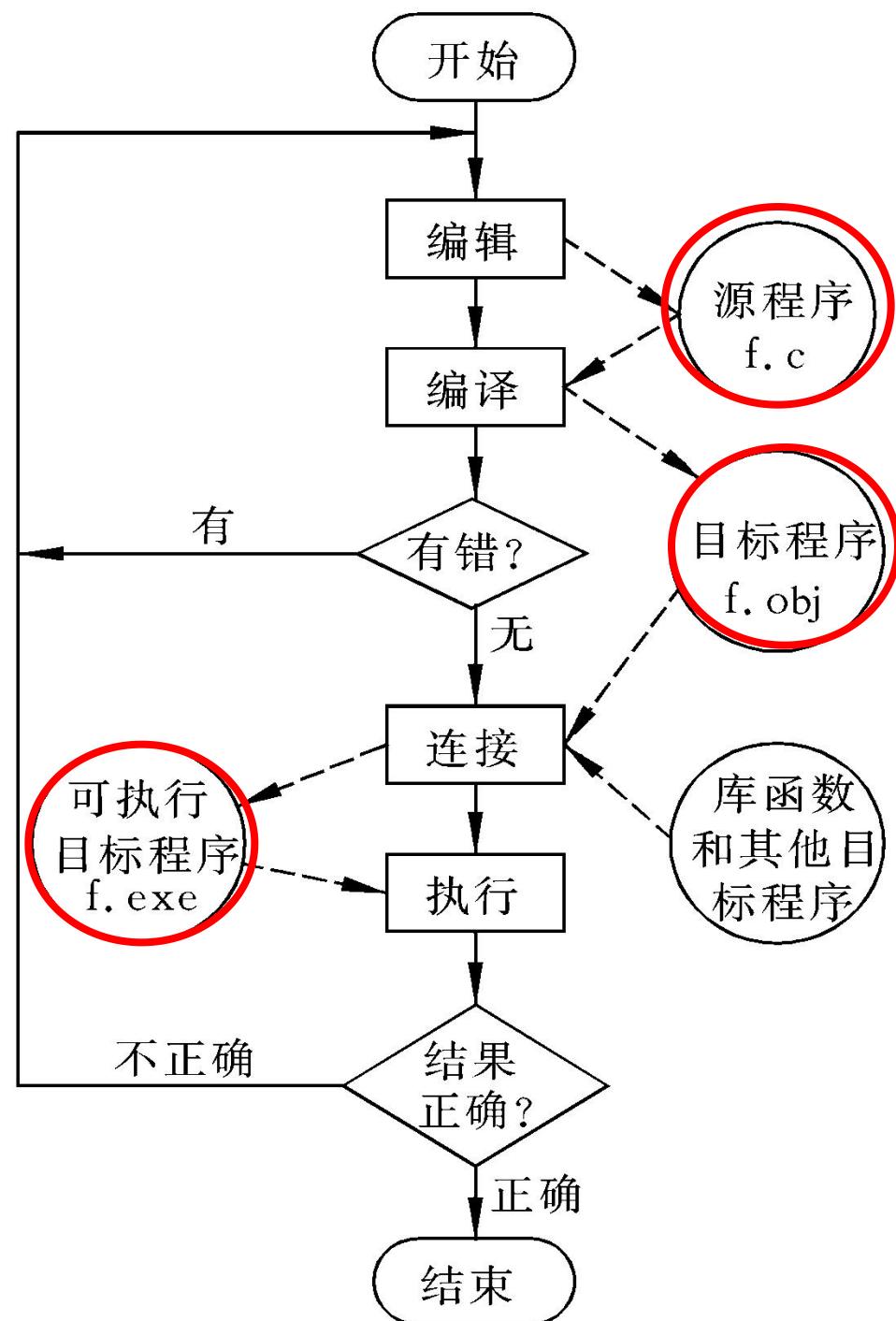
§1-3 简单的C语言程序介绍

- (3) 程序总是从main函数开始执行,与main函数的位置无关。
- (4) 程序书写格式自由,一行内可以写几个语句,一个语句可以分写在多行上。
- (5) 每个语句和数据声明最后必须有一个分号。

§1-4 运行C程序的步骤

一、运行C程序的步骤

- 上机输入与编辑源程序
- 对源程序进行编译
- 与库函数连接
- 运行目标程序



§1-4 运行C程序的步骤和方法

请养成良好的debug习惯！

写到有对不确定的语句，debug、test完毕，

再继续写下面的语句；

不要写一堆有很多不确定的语句，最后一块

debug。很多error message混在一起，会很

混乱。



VC++ 6.0 使用教程



用VC编写控制台程序

- Microsoft Visual C++（简称VC）是微软推出的一款基于其Windows系统C++编译器，将“高级语言”翻译为“机器语言（低级语言）”的程序。VC是一个功能强大的可视化软件开发工具。



什么是控制台程序？

- 控制台程序(**Win32 Console application**)，就是能够运行在**MS-DOS**环境中的程序。
- 控制台程序是为了兼容**DOS**程序而设立的，通常没有可视化的界面，只是通过字符串来显示或者监控程序，常常被应用在测试、监控等用途。



什么是控制台程序？

- 一个最简单的控制台程序如下：

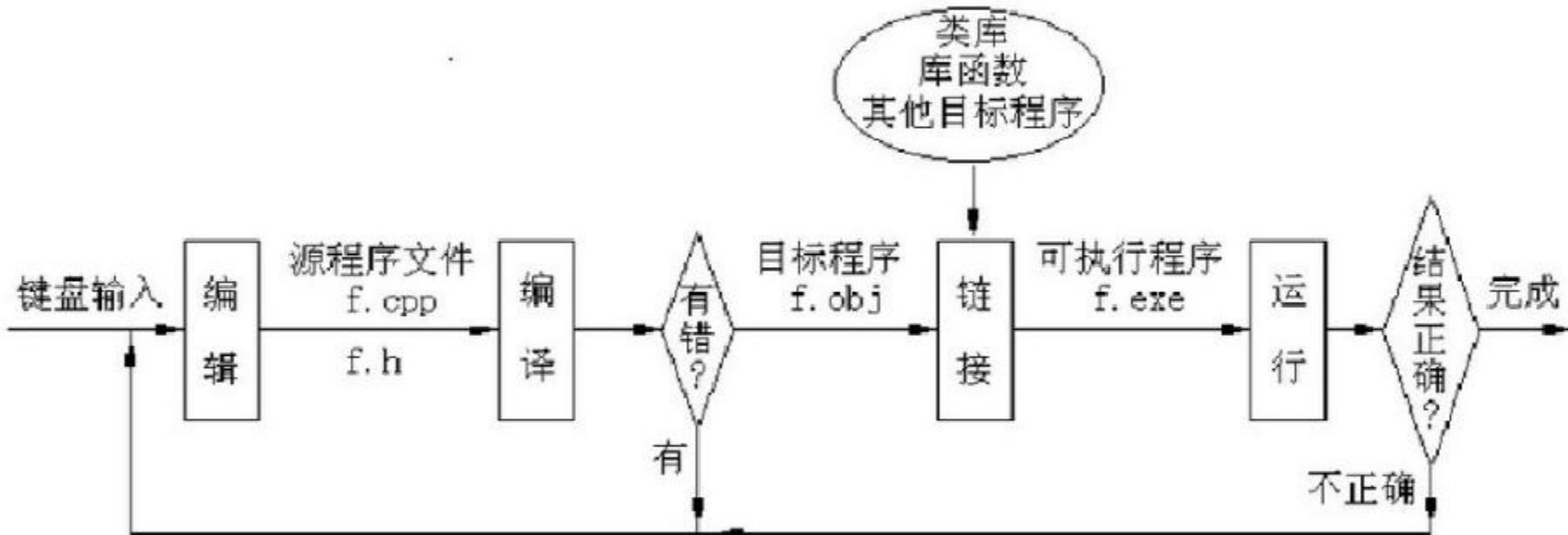
```
/* | D:\mywork\Debug\helloworld.exe
Hello, World!
Press any key to continue.
*/
#include <iostream>
int main()
{
    std::cout << "Hello, World!" << std::endl;
    std::cout << "Press any key to continue..." << std::endl;
}
```

这就是控制台窗口

图中显示的黑色窗口称为**控制台窗口**，程序的输入、输出均在这个窗口中进行。



C程序开发过程



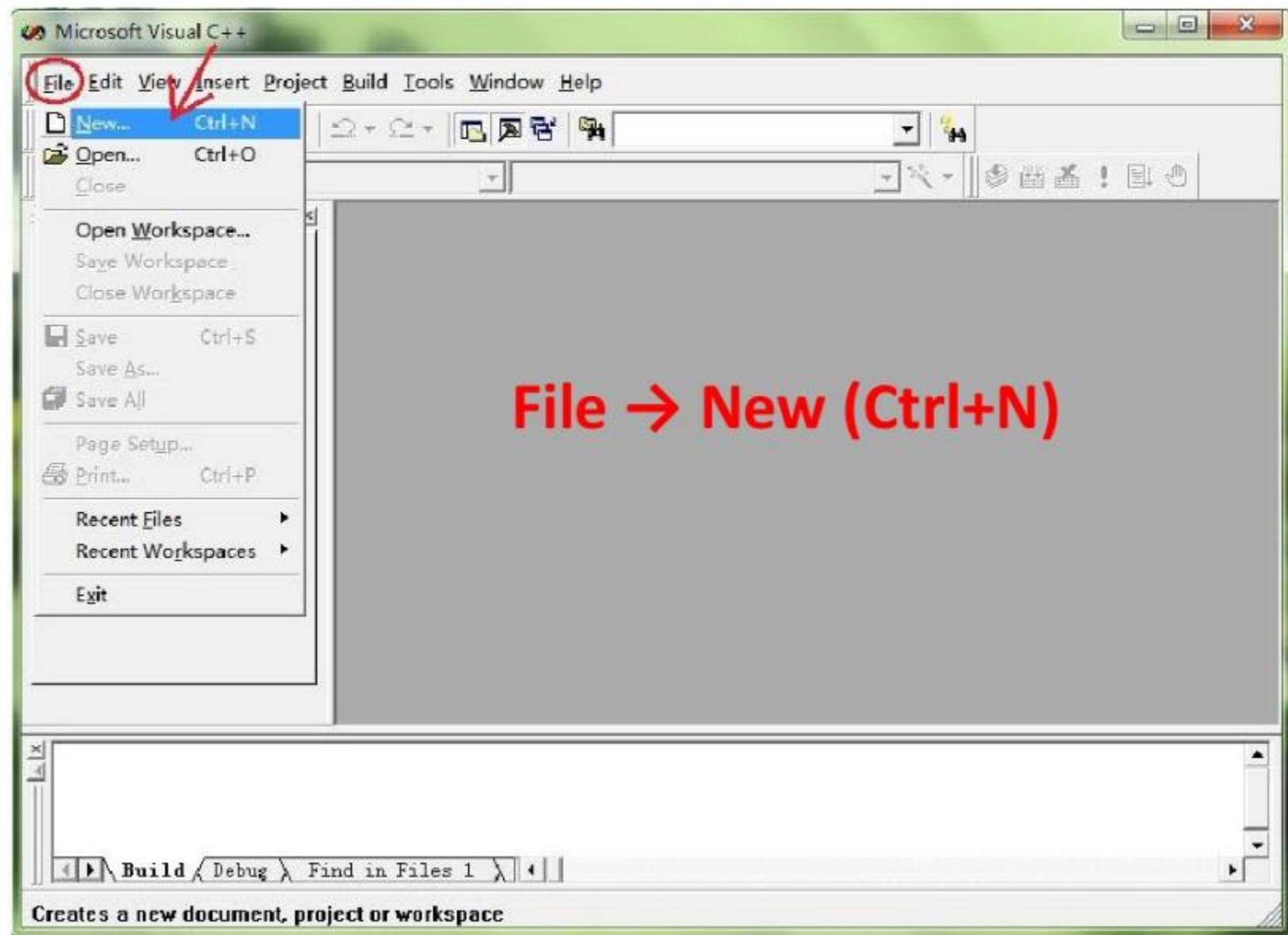


打开 VC++6.0



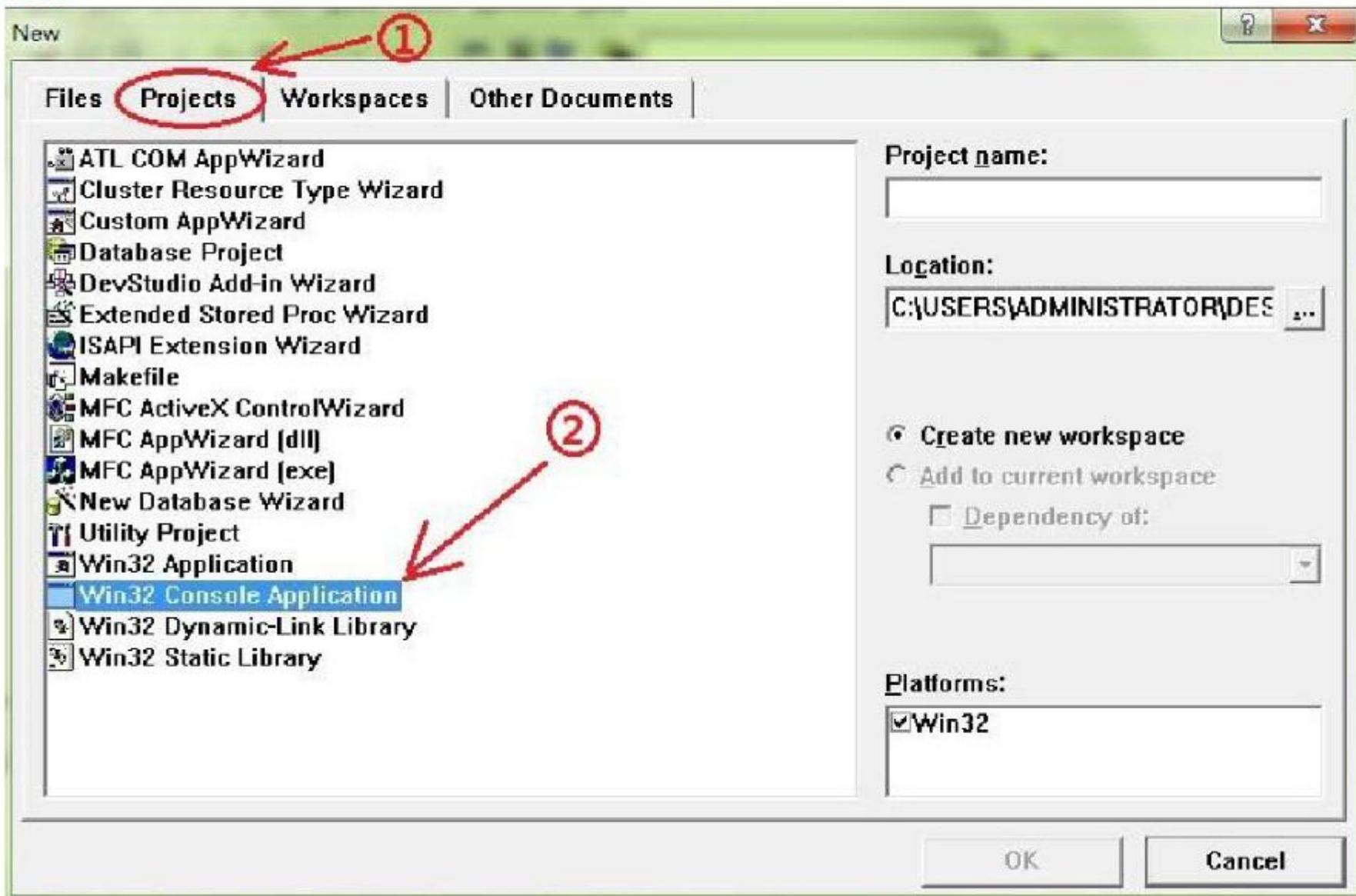


新建



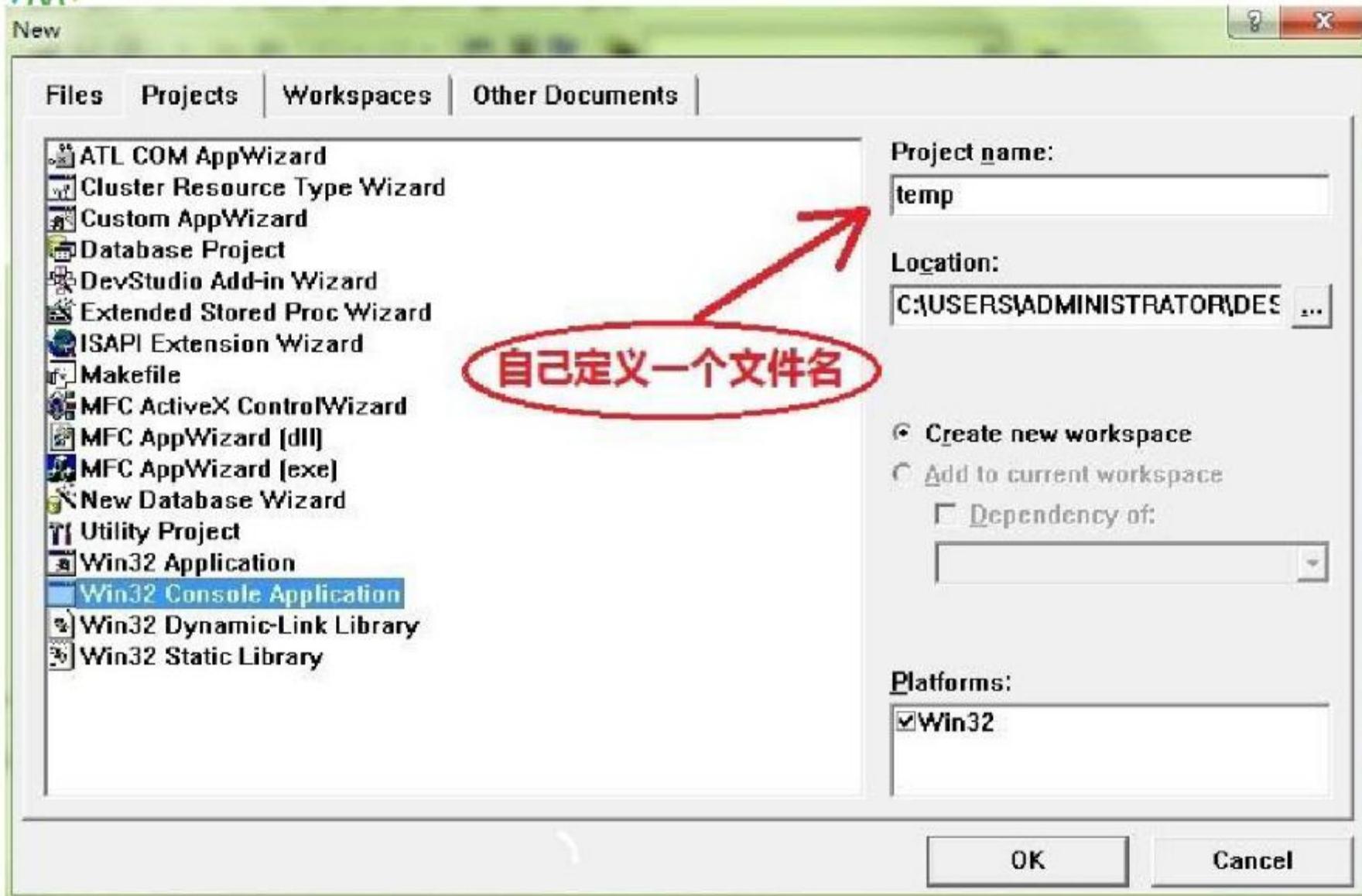


创建一个新的项目文件



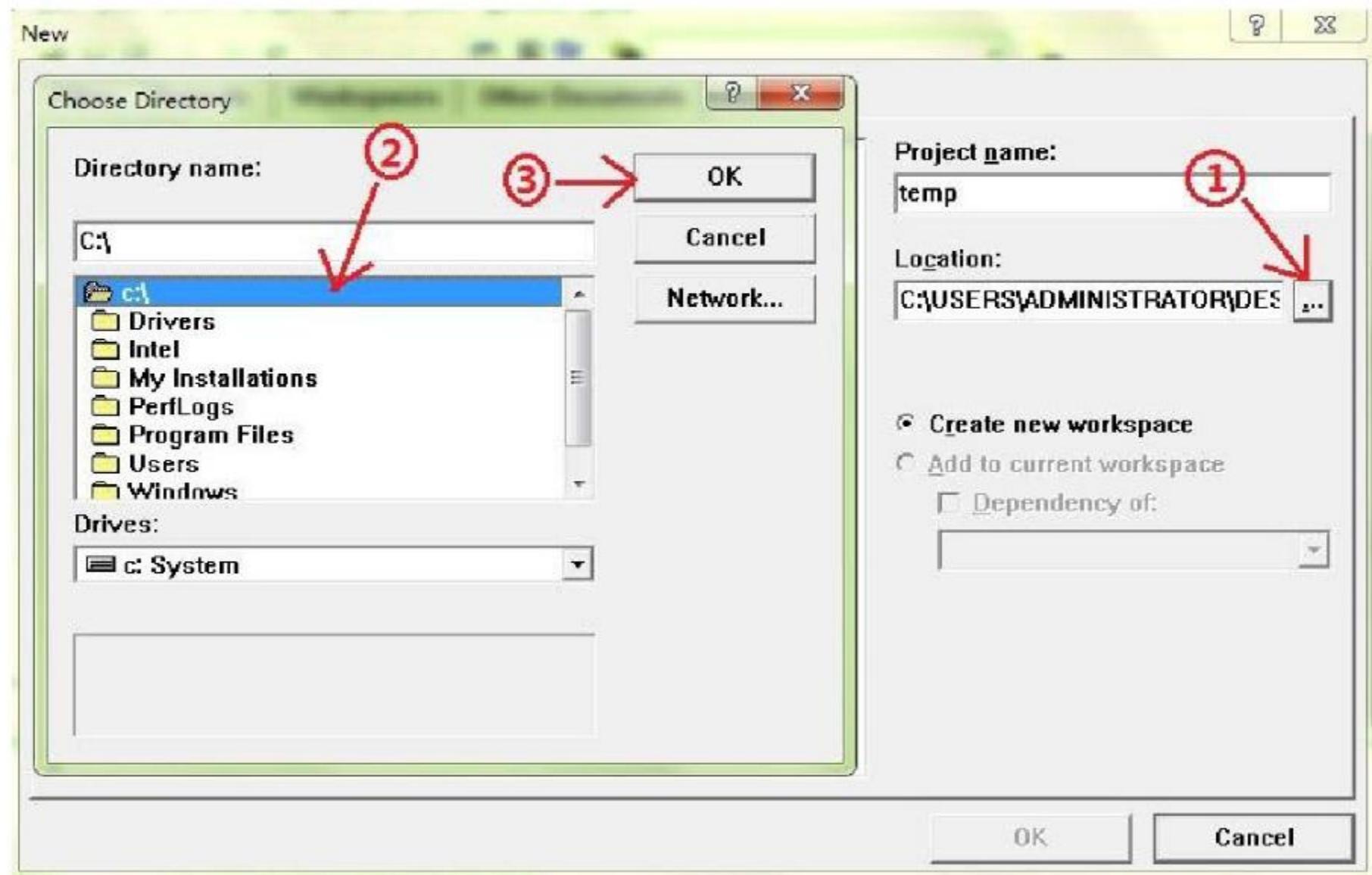


新建工程命名





修改工程存放路径





创建一个空工程

Win32 Console Application - Step 1 of 1

What kind of Console Application do you want to create?

An empty project.

A simple application.

A "Hello, World!!" application.

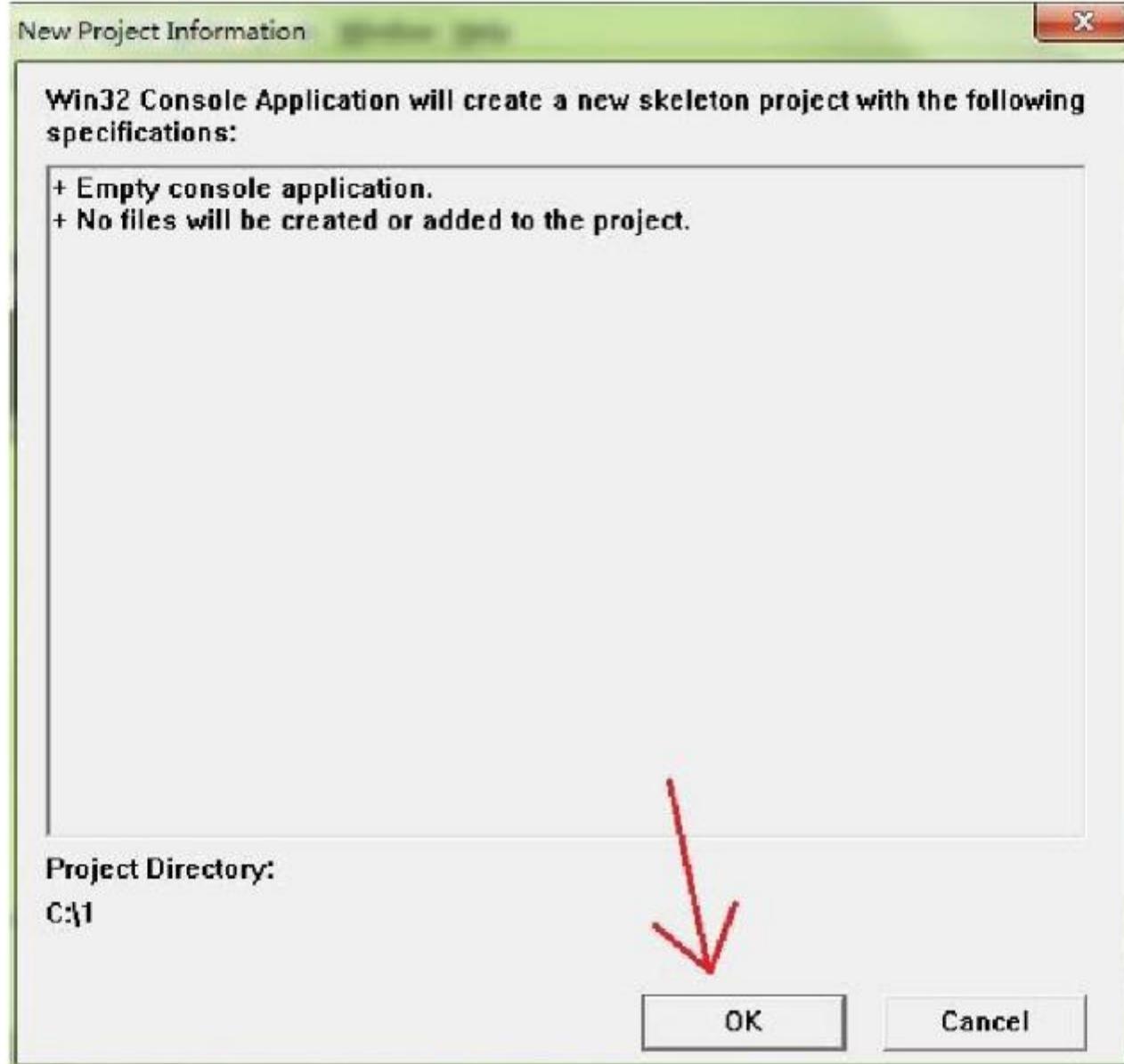
An application that supports MFC.



< Back Next > Finish Cancel

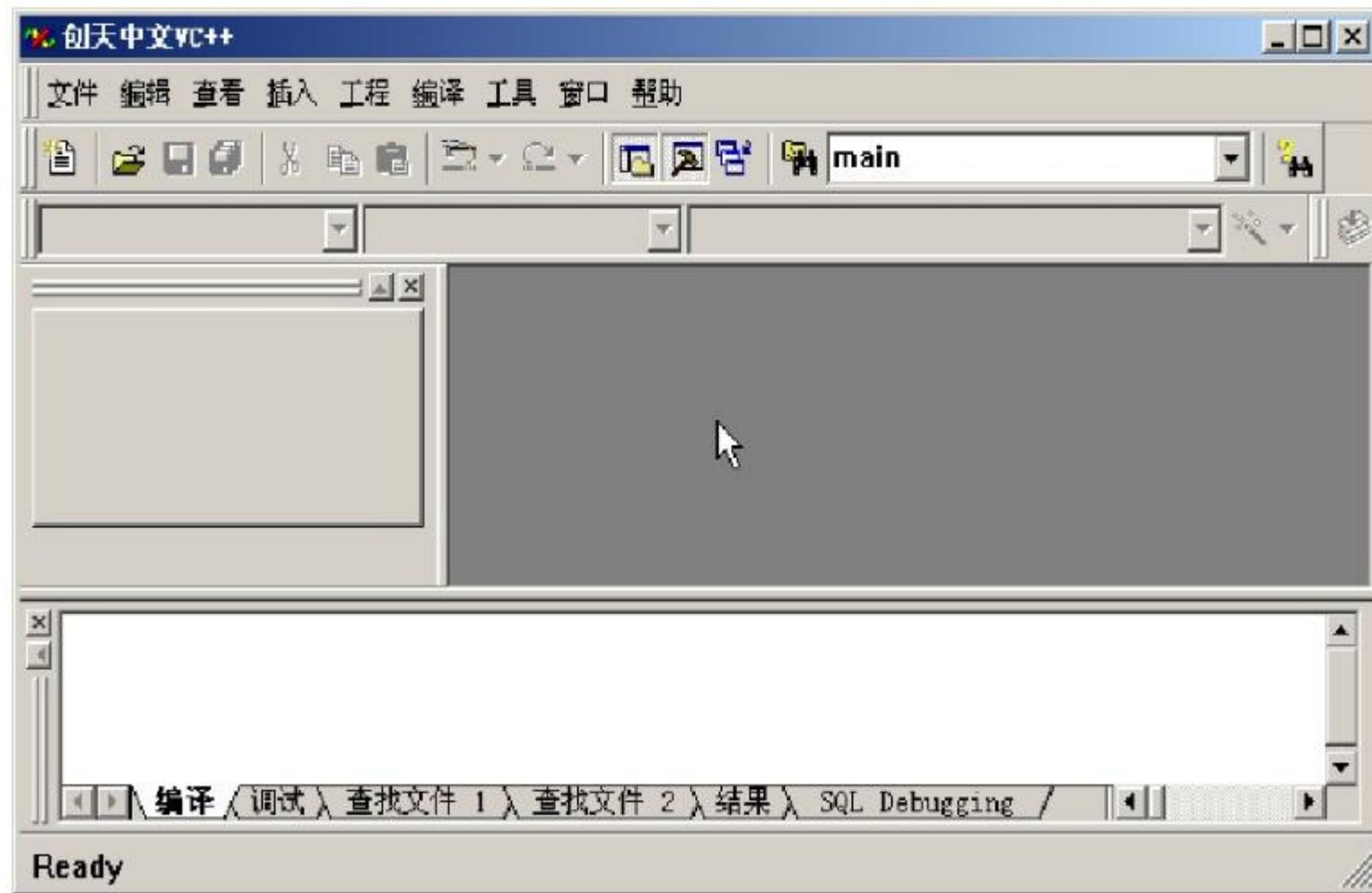


生成一个空工程



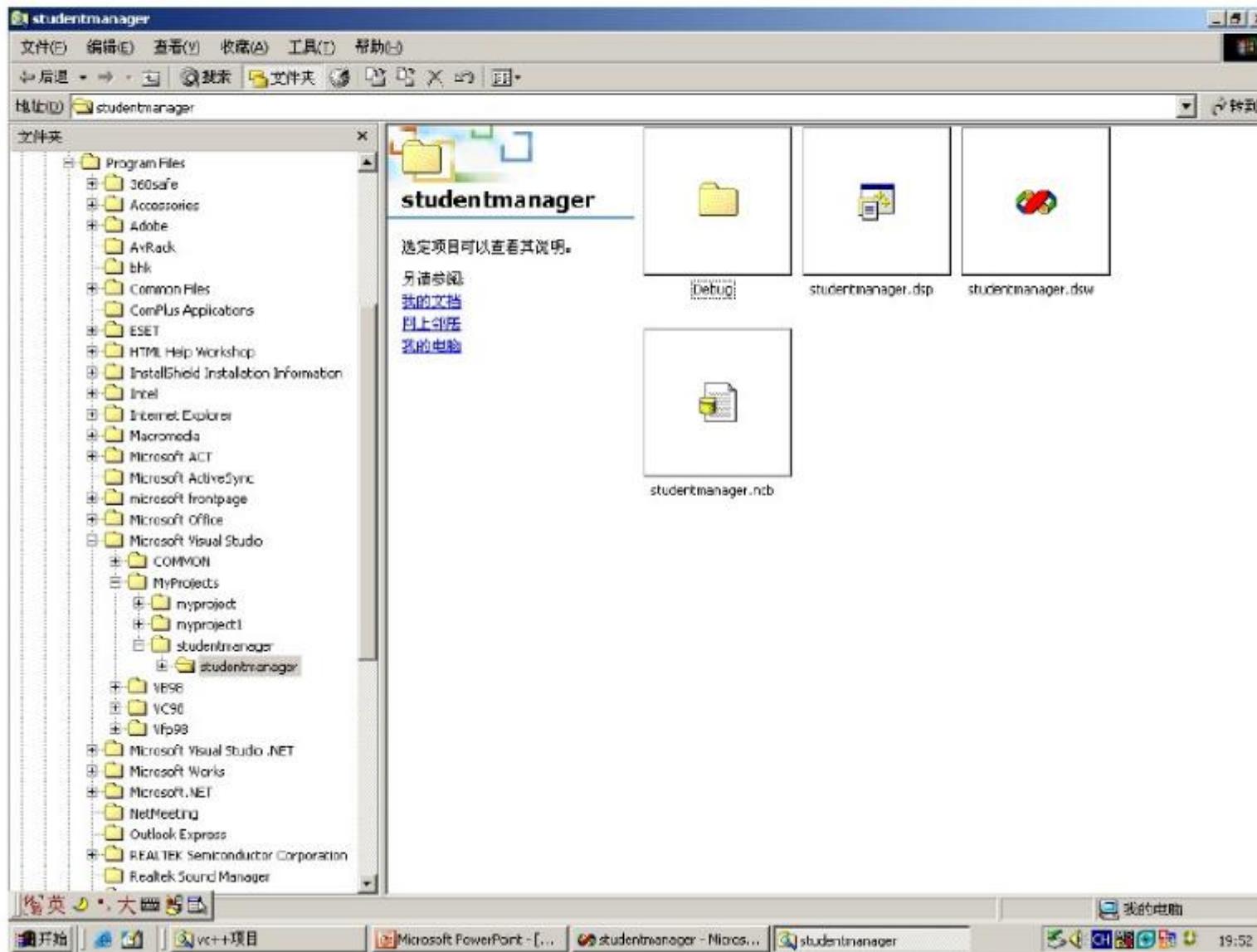


VC++6.0的编辑窗口



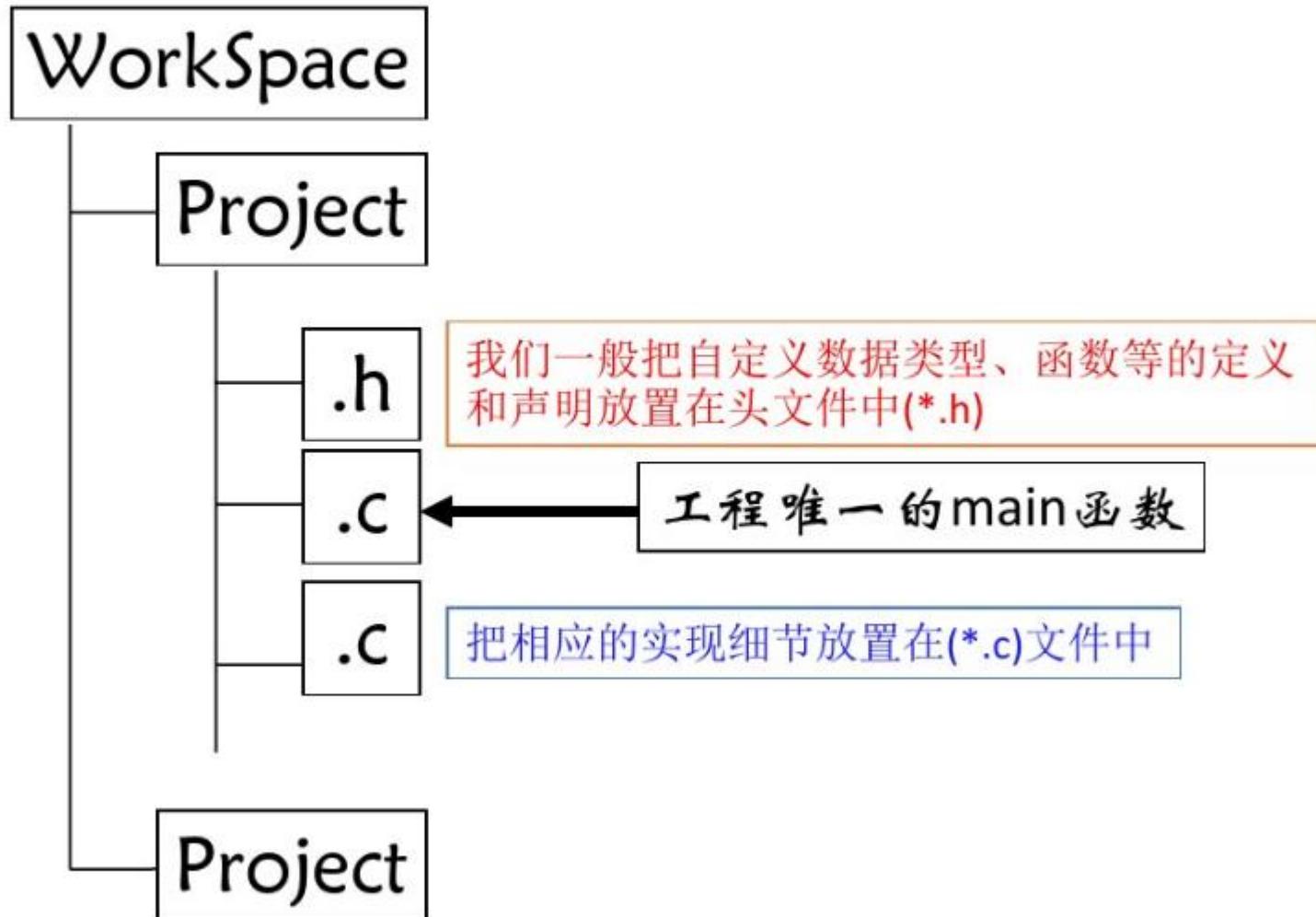


自动生成的文件和文件夹





工程管理结构



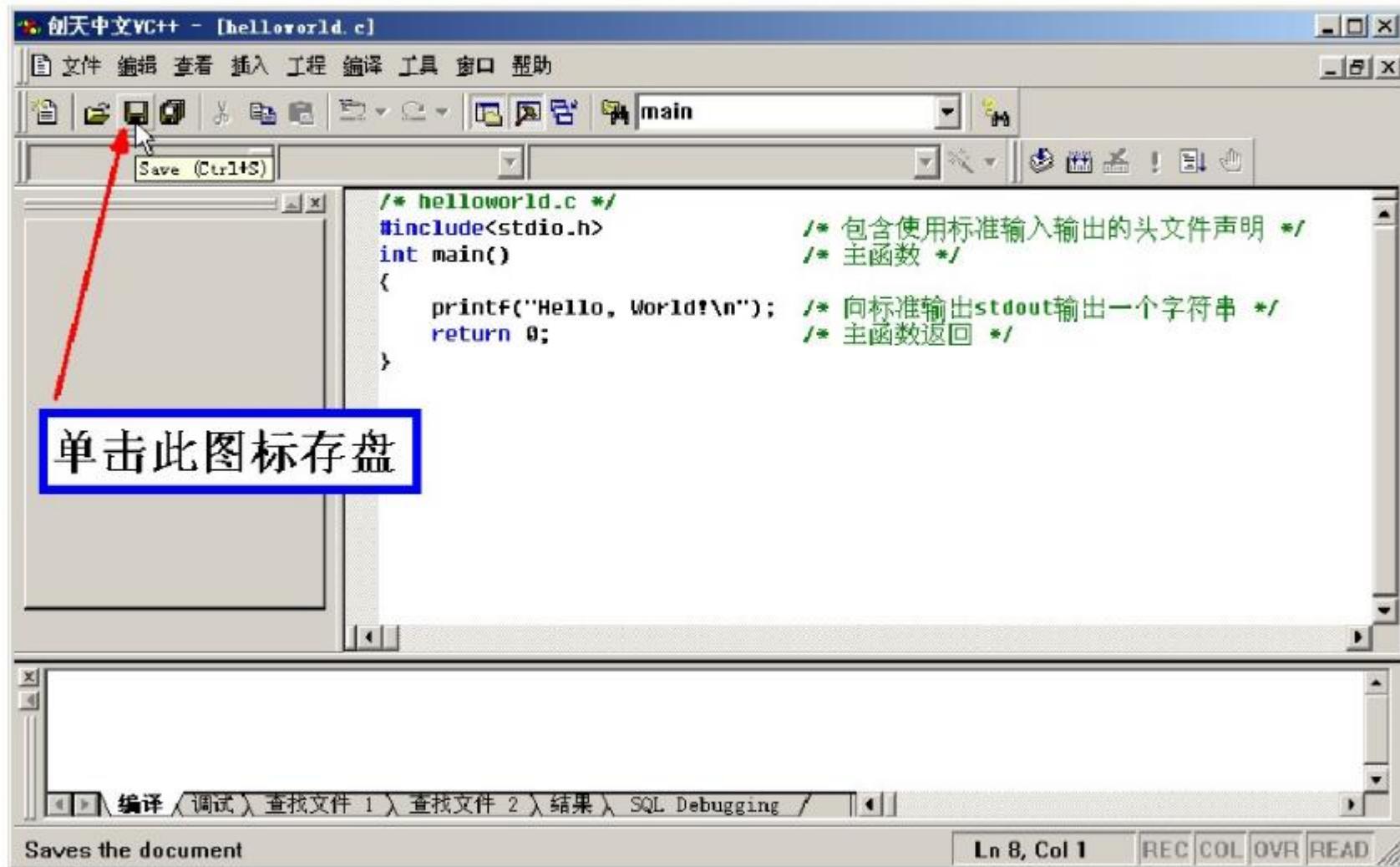


新建C源程序文件



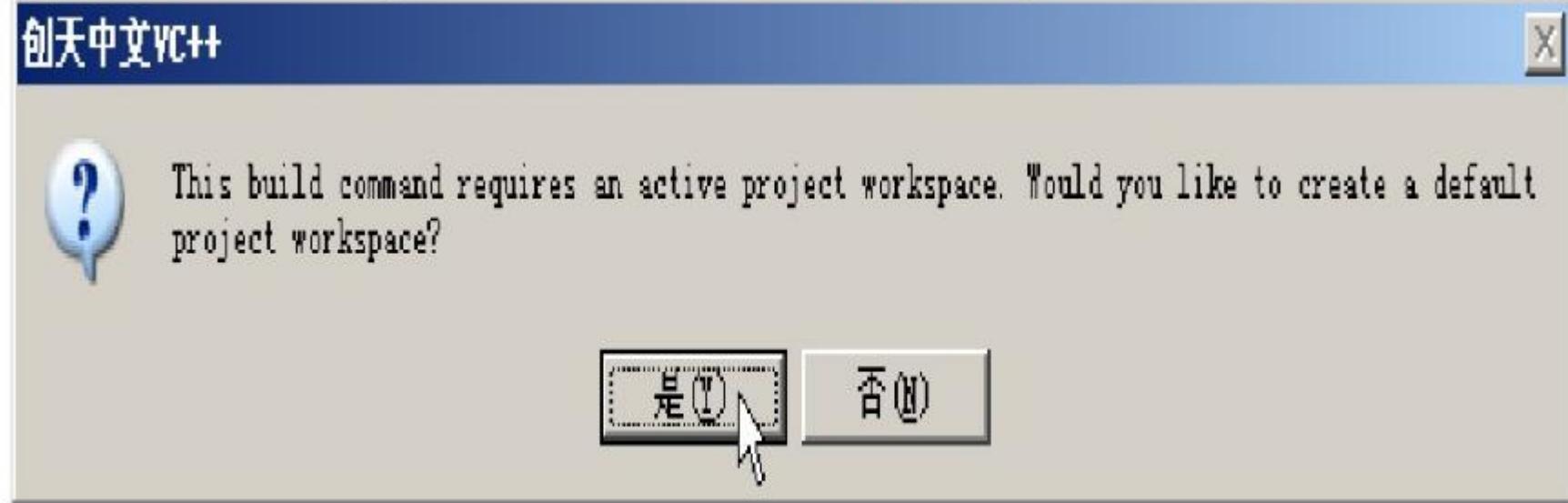
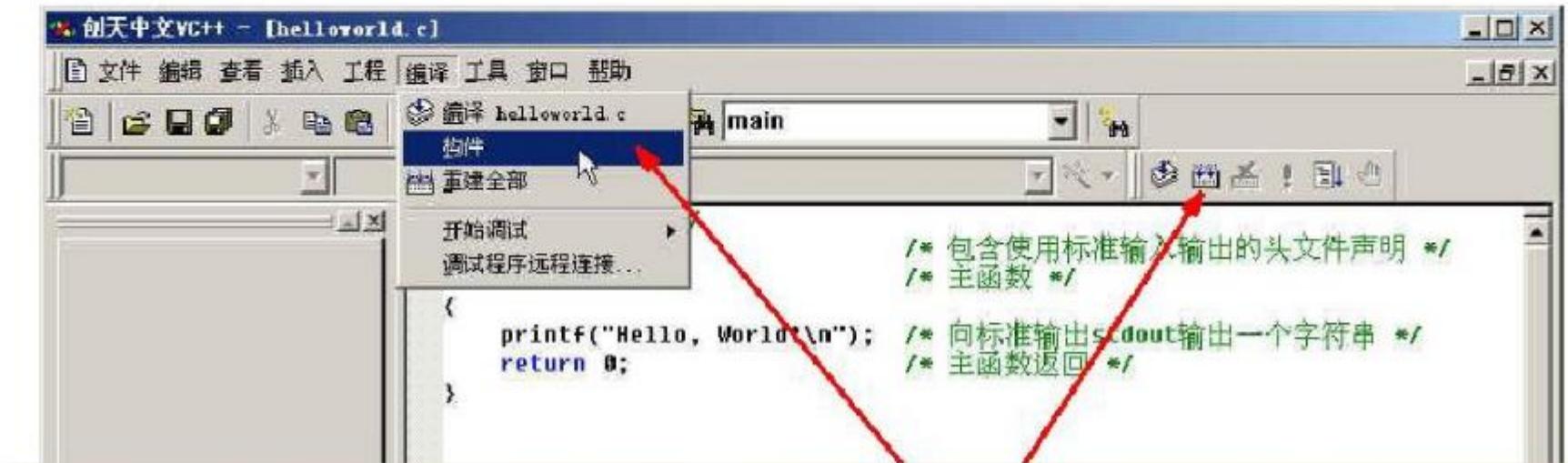


编辑文件





编译源程序 更简单可按F7键编译





编译源程序

helloworld - 创天中文VC++ - [helloworld.c]

文件 编辑 查看 插入 工程 编译 工具 窗口 帮助

[Globals] [All global members] main

helloworld classes

```
/* helloworld.c */  
#include<stdio.h>  
int main()  
{  
    printf("Hello, World!\n"); /* 包含使用标准输入输出的头文件声明 */  
    /* 主函数 */  
    /* 向标准输出stdout输出一个字符串 */  
    /* 主函数返回 */  
    return 0;  
}
```

哈哈，编译成功了！

helloworld.exe - 0 error(s), 0 warning(s)

编译 调试 查找文件 1 查找文件 2 结果 SQL Debugging /

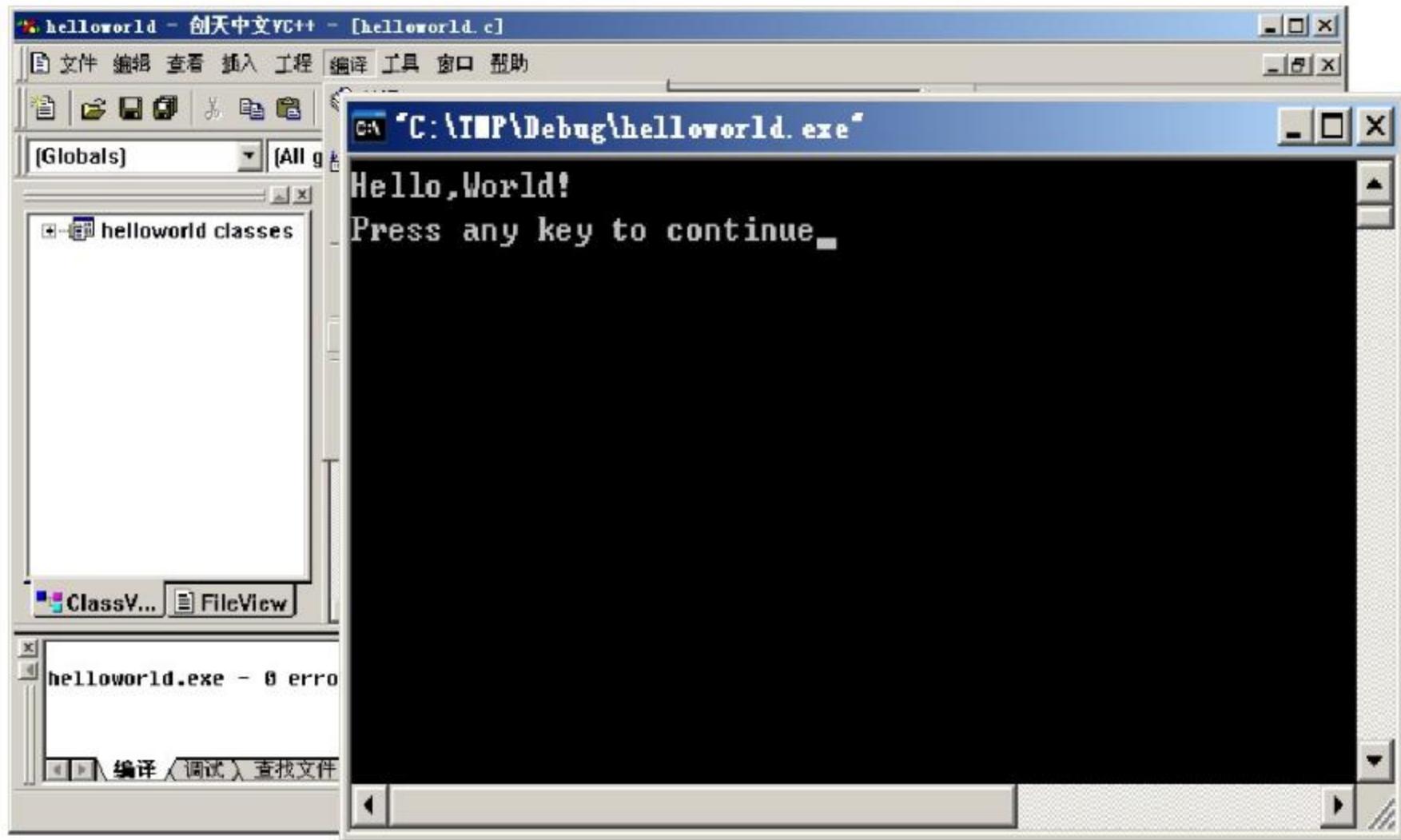
Ln 1, Col 14 REC COL OVR READ

```
/* helloworld.c */  
#include<stdio.h>  
int main()  
{  
    printf("Hello, World!\n"); /* 包含使用标准输入输出的头文件声明 */  
    /* 主函数 */  
    /* 向标准输出stdout输出一个字符串 */  
    /* 主函数返回 */  
    return 0;  
}
```



运行程序

可按Ctrl+F5运行！





断点调试程序

example02 - Microsoft Visual C++ [break] - [example02.cpp]

File Edit View Insert Project Debug Tools Window Help

[Globals] [All global members] main

```
#include<stdio.h>
void main()
{
    int i,sum;
    float multiple;
    sum = 0;
    multiple = 1.0;
    i = 1;
    while(i<=10)
    {
        sum = sum+i;
        multiple = multiple*i;
        i = i+1;
    }
    printf("sum = %d\nmultiple = %f",sum,multiple);
}
```

Context: main()

Name	Value
sum+i	CXX00
multiple*i	CXX00

Auto Locals this / Watch1 Watch2 Watch3 Watch4

Ready



断点调试程序 F9

example02 - Microsoft Visual C++ [break] - [exmaple02.cpp]

File Edit View Insert Project Debug Tools Window Help

[Globals] [All global members] main

```
#include<stdio.h>
void main()
{
    int i,sum;
    float multiple;
    sum = 0;
    multiple = 1.0;
    i = 1;
    while(i<-10)
    {
        sum = sum+i;
        multiple = multiple*i;
        i = i+1;
    }
    printf("sum = %d\nmultiple = %f",sum,multiple);
}
```

Context: main()

Name	Value
i	1
multiple	1.00000
sum	1

Auto Locals this

Name Value

Name	Value
sum+i	2
multiple*i	1.00000

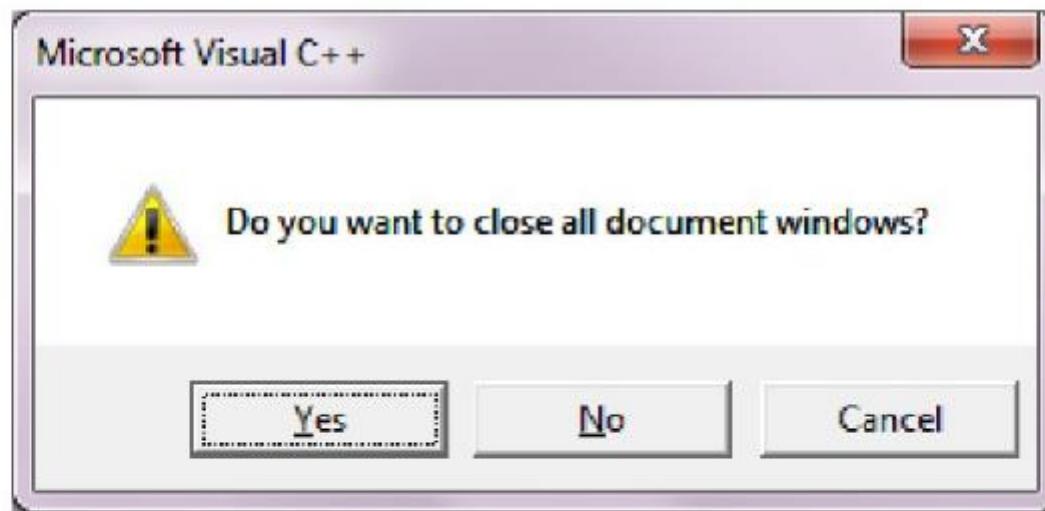
Watch1 Watch2 Watch3 Watch4

Ready



关闭文件和工作区

选择菜单“File\Close WorkSpace”，关闭当前的工作区和所有打开的文件。



本课程大体规划：

C语言概述、程序的灵魂——算法、数据类型、运算符与表达式、最简单的程序设计 **1周**

选择结构程序设计、循环控制 **1周**

数组 **1周**

函数、预处理命令 **1周**

指针 **2周**

结构体和共用体 **1周**

位运算、文件操作与总结 **1周**

如果有时间：并行计算 **1周**

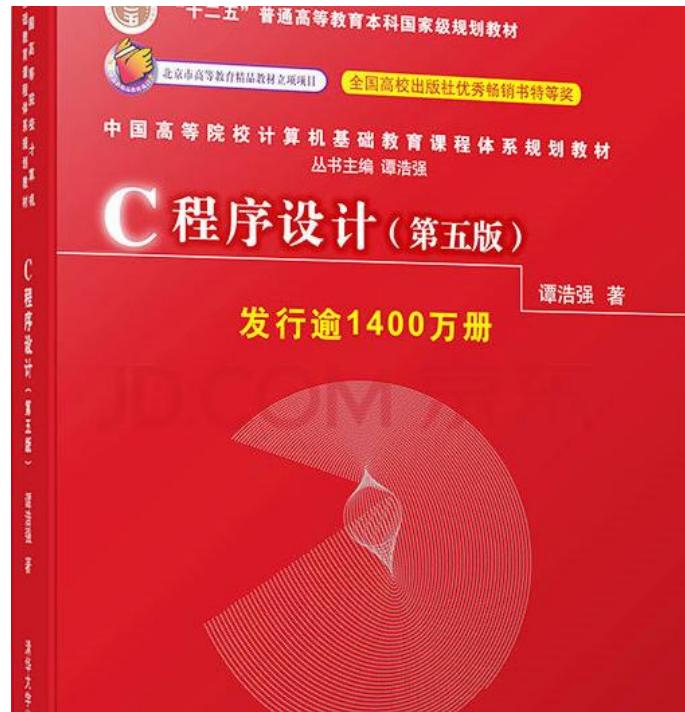
教材：

谭浩强《C程序设计(第五版)》

邮箱：

sysuc123456@163.com

密码：**sysuc1234567**



第二章

程序的灵魂--算法

一个程序包括两个方面：

1. 对数据的描述：数据结构(data structure)
2. 对操作的描述：算法(algorithm)

著名计算机科学家沃思提出一个公式：
U U 数据结构 + 算法 = 程序□

更完整应该是：

数据结构 + 算法 + 程序设计方法 + 语言工具

§2.1 算法的概念

对同一个问题，可有不同的解题方法和步骤

例：求 $\sum_{n=1}^{100} n$

- 方法1： $1+2, +3, +4, \dots, +100$
- 方法2： $100+(1+99)+(2+98)+\dots+(49+51)+50$

$$= 100 + 49 \times 100 + 50$$

§2.1 算法的概念

计算机算法可分为两大类别：

- 数值运算算法：例如寻找方程的根；求微分、积分；解微分方程；等等。
- 非数值运算：十分广泛，常见于事务管理领域，例如图书检索、人事管理、行车调度管理；等等。

例2.1 对一个大于或等于3的正整数，判断它是不是一个素数。

分析：判断一个数 $n(n \geq 3)$ 是否素数的方法：

n 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。

算法如下：

S1：输入n的值H

S2： $i=2$ H (i作为除数)

S3：n被i除，得余数rH

S4：如果 $r=0$ ，表示n能被i整除，则打印n“不是素数”，算法结束。否则执行S5H

S5： $i=i+1$ H

S6：如果 $i \leq n-1$ ，返回S3。否则打印 n “是素数”。然后结束。H

实际上，n不必被2到($n-1$)的整数除，只需被2到根号n之间的整数除即可。

§2.3 用流程图表示算法

美国国家标准协会ANSI(American National Standard Institute)规定常用的流程图：



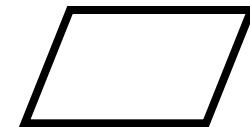
起止框



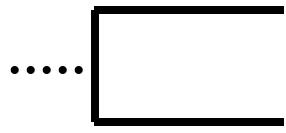
判断框



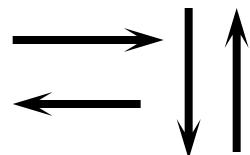
处理框



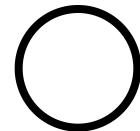
输入/输出框



注释框

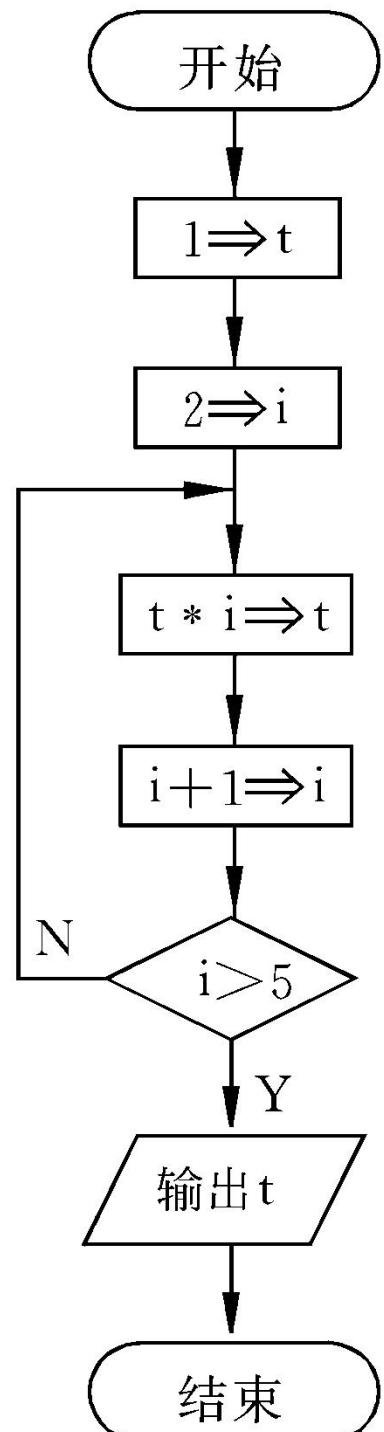


流向线

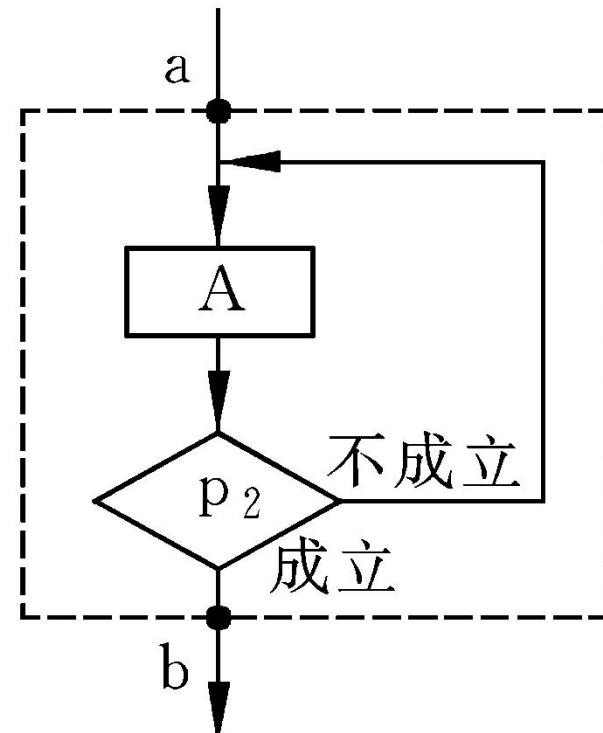
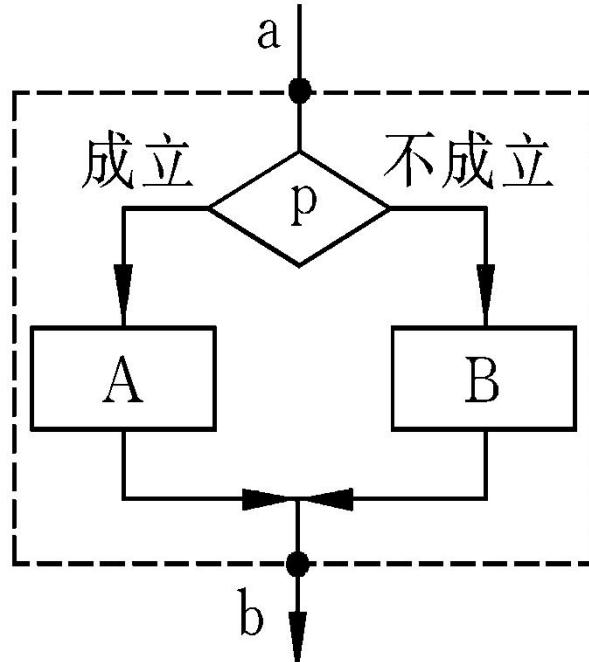
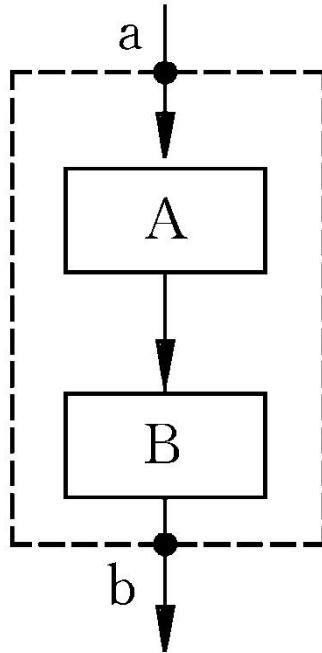


连接点

例2.2 将求 $5!$ 的算法用流程图表示



三种基本结构的图示：



例 2.3 将例2.2表示的算法（求5!）用C语言表示。

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

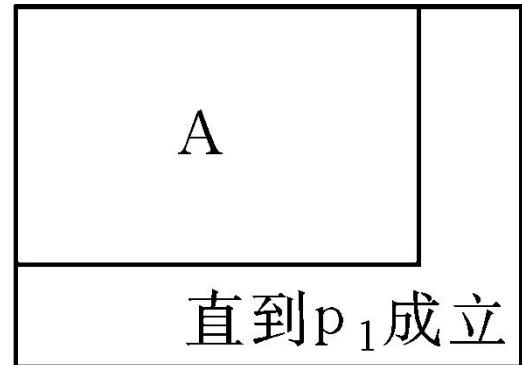
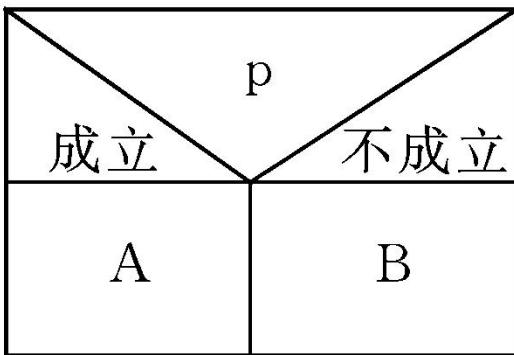
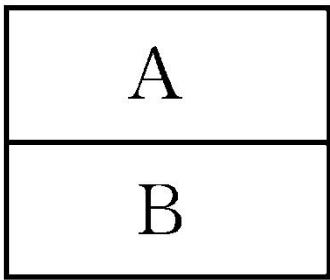
§2.4.4 用N--S流程图表示算法

1973年美国学者I.Nassi和B.Shneiderman提出了一种新的流程图形式：

全部算法写在一个矩形框内，在该框内还可以包含其它的从属于它的框，或者说，由一些基本的框组成一个大的框。

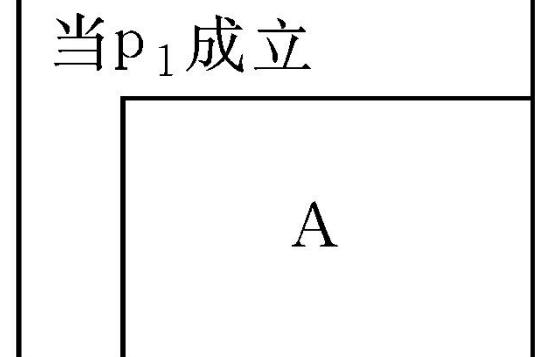
这种流程图又称N--S结构化流程图。

N--S流程图用以下的流程图符号：



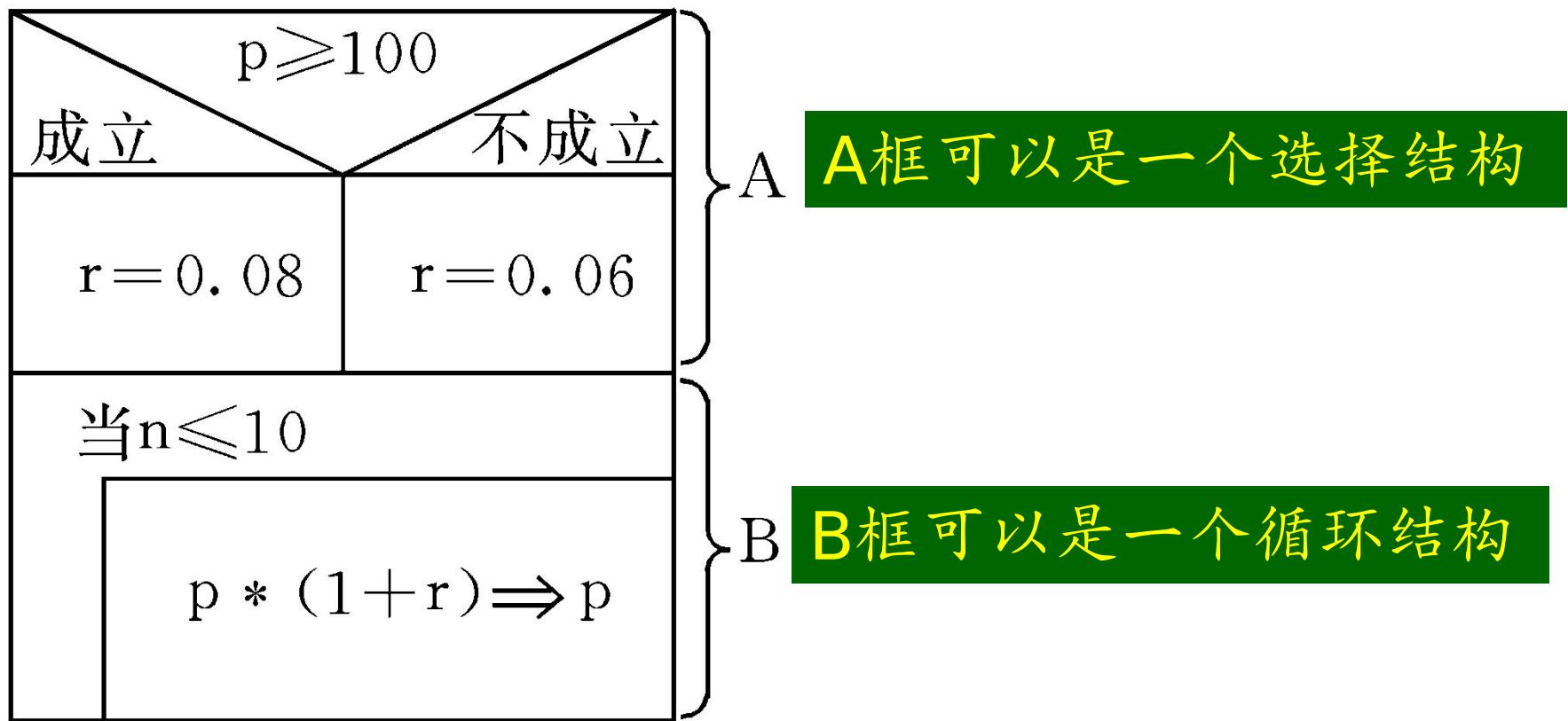
(1)顺序结构○

(2)选择结构○

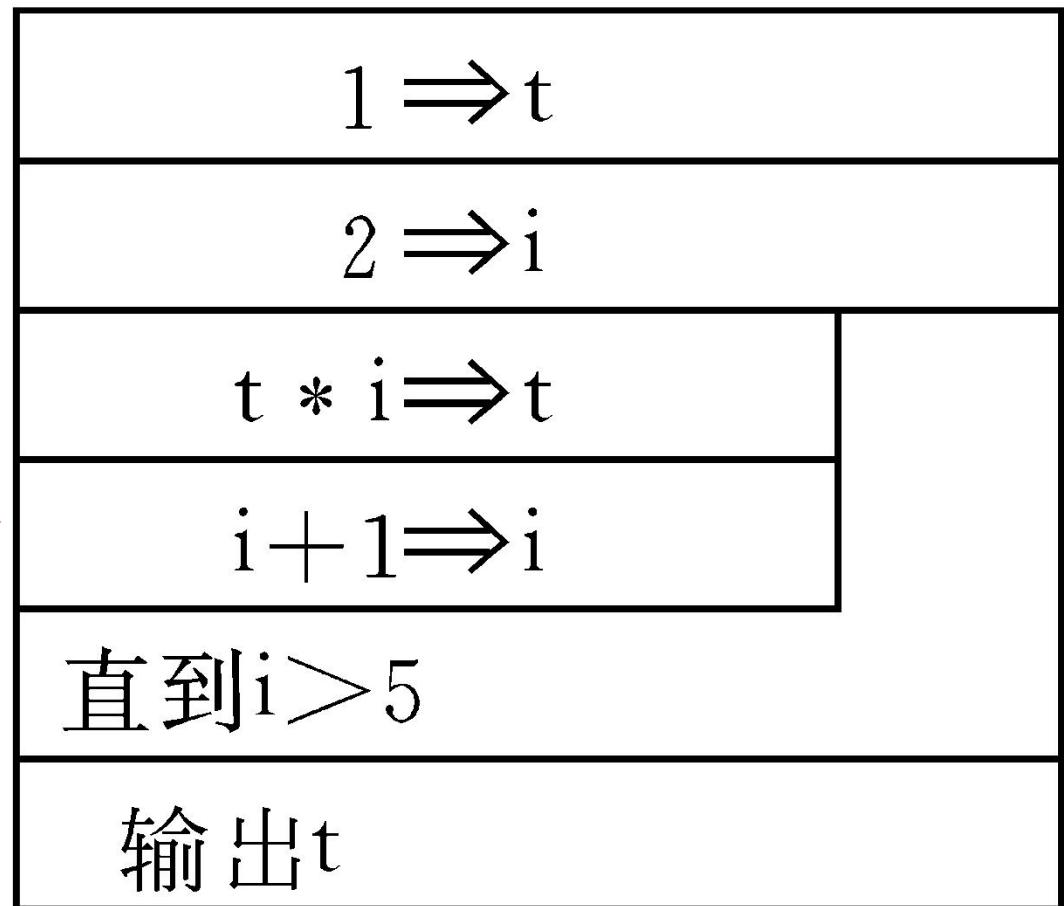
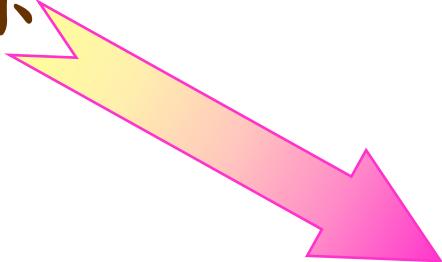


(3)循环结构○

用三种N--S流程图中的基本框，可以组成复杂的N--S流程图。图中的A框或B框，可以是一个简单的操作，也可以是三个基本结构之一。



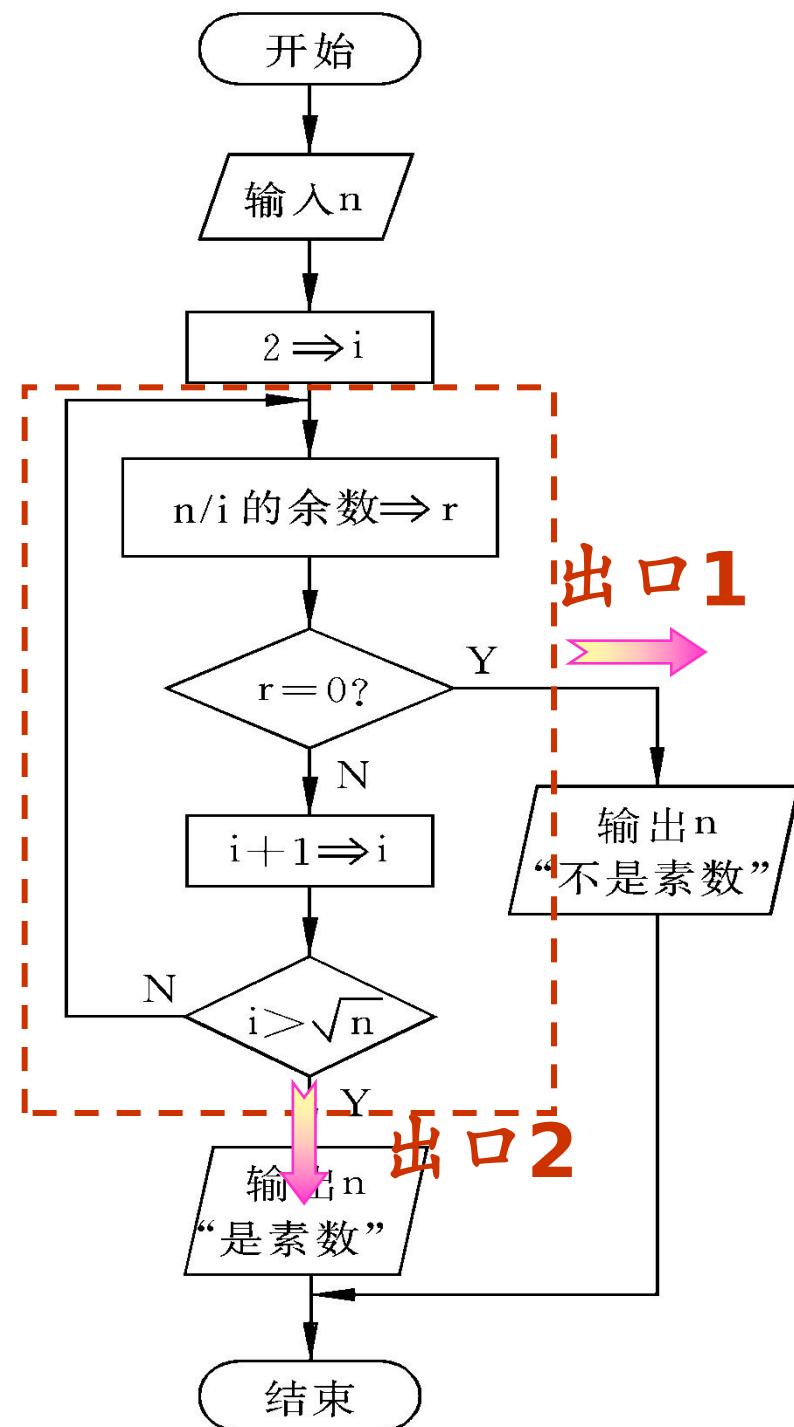
例2.11 将例
2.1的求 $5!$ 算
法用N--S图
表示



例2.15 将例2.5判断素数的算法用N--S流程图表示

传统流程图分析：

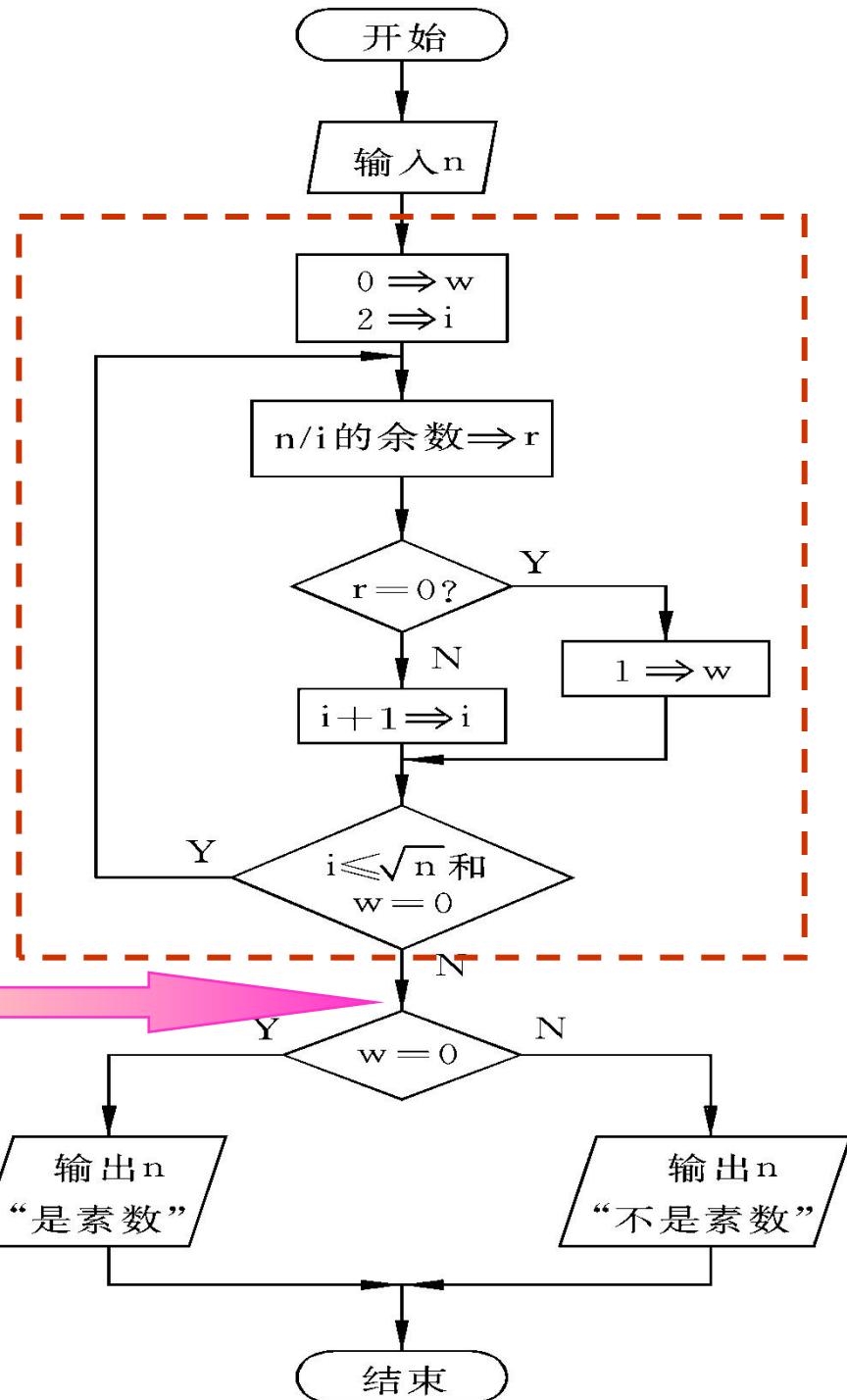
此图不符合基本结构特点！
由于不能分解为三种基本结构，就无法直接用N--S流程图的三种基本结构的符号来表示。因此，应当先作必要的变换。



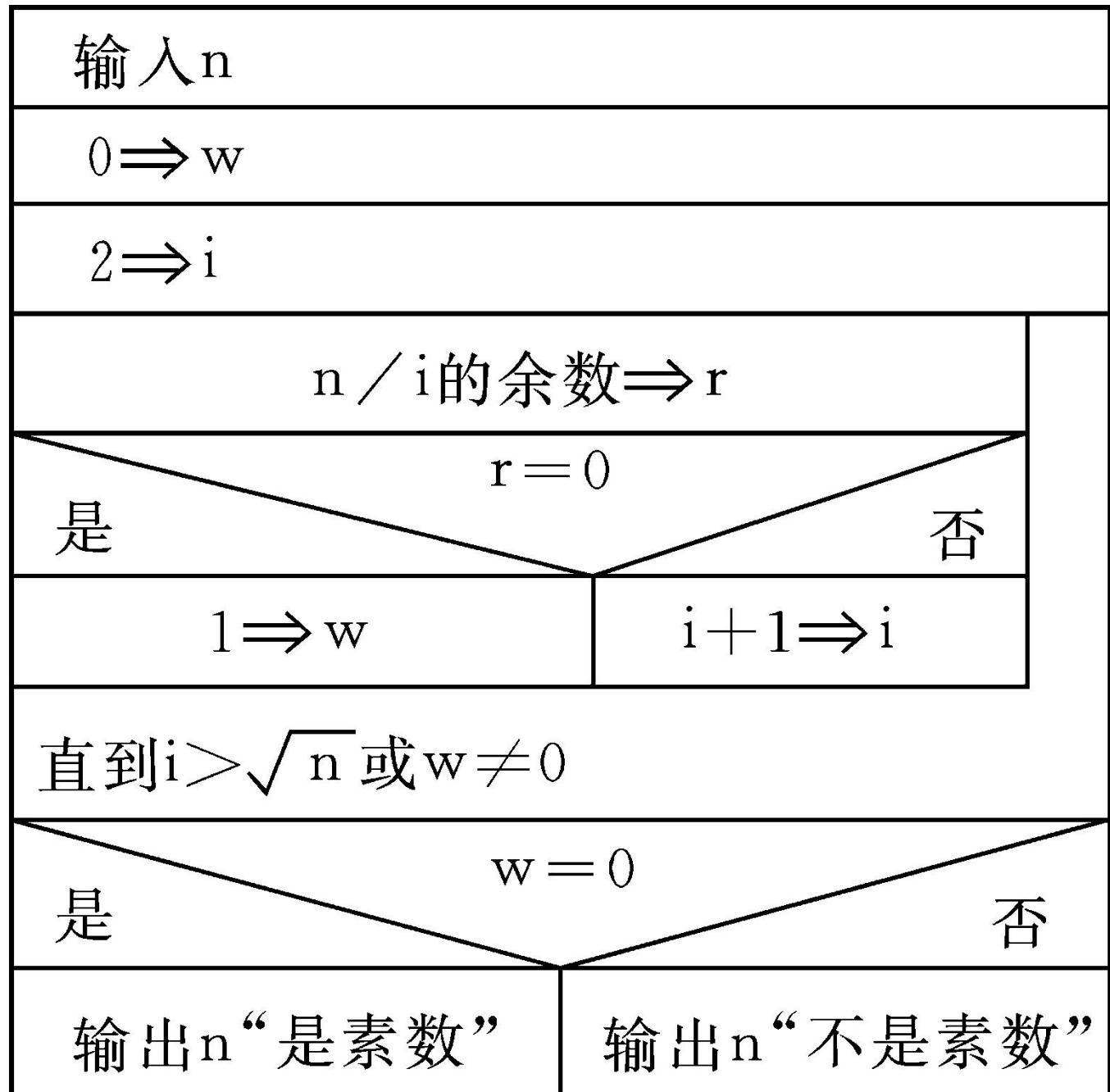
例2.15 将例2.5判断素数的算法用N--S流程图表示

传统流程图变换为：

一个出口



用
N-S
流程图表示..



N-S图表示算法的优点

- 比文字描述直观、形象、易于理解；
比传统流程图紧凑易画。尤其是它废除了流程线，整个算法结构是由各个基本结构按顺序组成的，N-S流程图中的上下顺序就是执行时的顺序。
- 用N-S图表示的算法都是结构化的算法，因为它不可能出现流程无规律的跳转，而只能自上而下地顺序执行。

- 补充作业：
 - 有笔记本的同学，自行安装 Visual Studio 20** !
 - 练习使用 Visual Studio 建立 project，将 ppt 中的示例程序编译、运行。
 - 尝试编写如下程序：
 - 计算 $1+2+3+\dots+1000?$
 - 计算 $1 - \frac{1}{2} + \frac{1}{3} - \frac{1}{4} + \frac{1}{5} \dots - \frac{1}{1000} ?$

- 作业（也适用以后每一章）
 - 自觉弄懂ppt中所有知识点！
 - 写程序测试所有语句
 - 例题弄明白
 - 完成教科书相应章节的习题！（强烈建议用自己的电脑）
 - 学号单号同学：1、3、5、...；双数同学：2、4、6、...
 - 问答题明白就行，不用做
 - 第2章所有人都只做第8题（画流程图，两种自己挑）
 - 编程题完成后把 代码 + 运行结果截图 电子版发给各班班长或学委，标明哪些题调试没有通过（不标明的扣分），命名为：

99_wangsicong_week1.cpp (99代表学号末后两位)

99_wangsicong_week1.png

各班汇总后，打包发到邮箱：sysuc123456@163.com

问一句：有没有不属于1-3班的同学？

建议：可用一个 **main** 函数把所有题写在一起，第一行打印姓名学号
多个{}完成多题。{}内的变量为局部变量，只在此{}内生效。

- **#include <stdio.h>**
- **void main()**
- {
- **printf("This is week 1 homework of xiaodongli, no 99.\n");**
- {
- **printf("hw 1.1\n");**
- **int i=1;**
- **printf("%d\n",i);**
- }
- {
- **printf("hw 1.2\n");**
- **int i,j;**
- **for(i=1,j=0;i<=100;i++)**
- **j+=i;**
- **printf("%d\n",j);**
- }
- **...**
- }

纪律

作业必须按时交，不许补交（迟交1周扣一半，2周0分）！

（做不出来的题目，要把代码、报错信息交上）

严禁抄袭。发现两个人都记0分（我们会仔细看代码）！

做题时不许看《教辅》的答案，更不许抄袭（发现记0分）

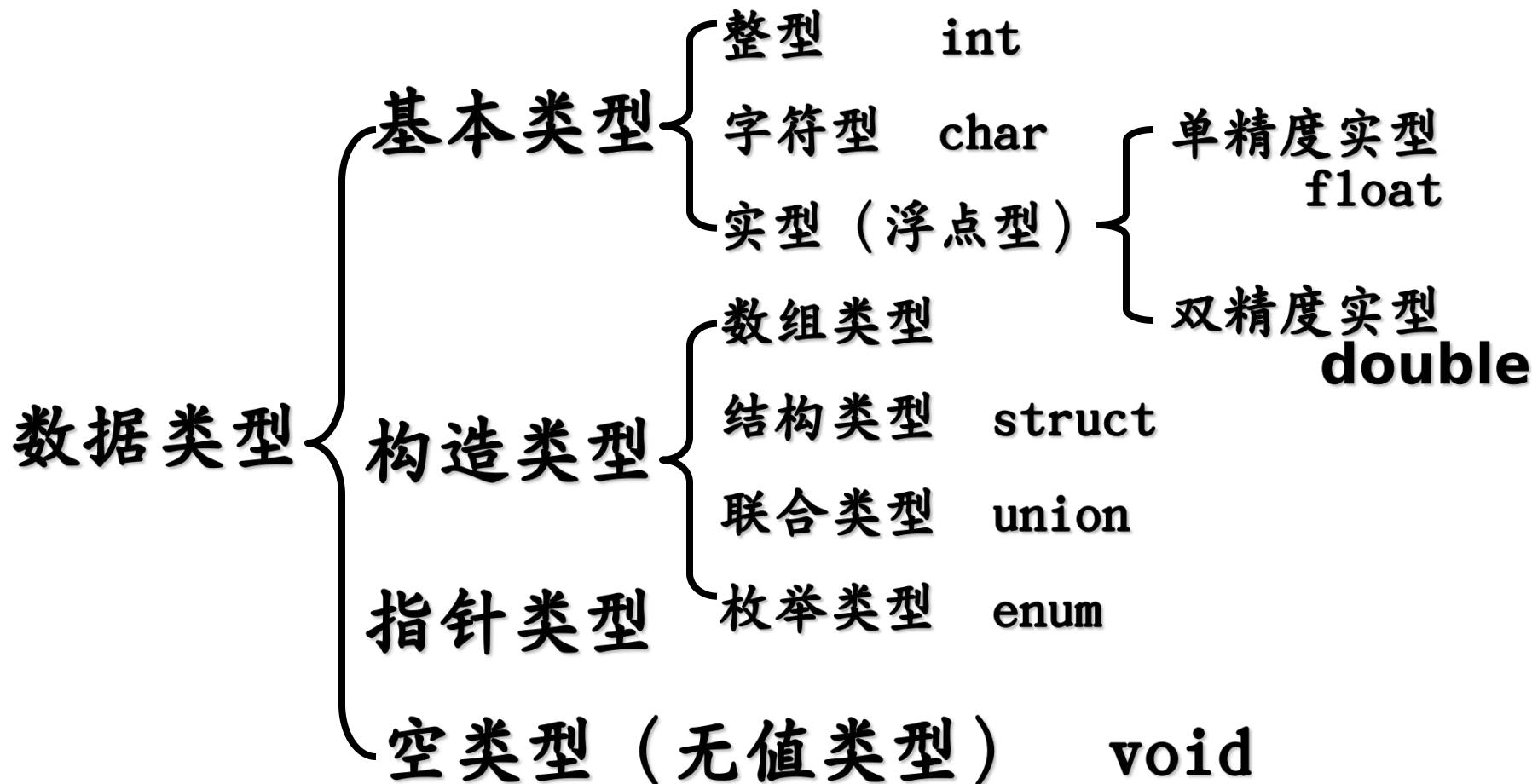
严禁伪造运行结果。助教会把代码运行测试，一旦发现记0分

第三章

数据类型、运算符与表达式

§ 3.1 C的数据类型

C语言提供了以下一些数据类型。



§3.2 常量与变量

- 程序运行过程中,值不能被改变的量称为常量
- 变量代表内存中具有特定属性的一个存储单元，它用来存放数据，这就是变量的值。在程序运行期间，这些值是可以改变的。
- 变量名实际上是以一个名字对应代表一个地址，在对程序编译连接时，由编译系统给每一个变量名分配对应的内存地址。
- 从变量中取值，实际上是通过变量名找到相应的内存地址，从该存储单元中读取数据。

§3.2 常量与变量

变量命名的规定：

字母、数字和下划线组成

第一个字符必须为字母或下划线。

例：sum , _total, month, Student_name ,
lotus_1_2_3 , BASIC, li_ling •

不合法的变量名：

M.D.John, ¥123,3D64,a>b

§3.2 常量与变量

注意：

- 区分大小写。
- 选择变量名时，应注意做到“见名知意”
- 要求对所有变量作类型声明，也就是“先声明，后使用”

§3.3 整型数据

(1) 整型数据在内存中的存放形式

如： int i; /* 定义为整型变量 */
i=10; /* 给i赋以整数10 */

- 十进制数10的二进制形式为1010。
- 数值是以补码(complement) 表示的。

§3.3 整型数据

类型	类型说明符	长度	数的范围 (不同编译器下不同)
基本型	int	2字节	-32768~32767
短整型	short	2字节	$-2^{15} \sim 2^{15}-1$
长整型	long	4字节	$-2^{31} \sim 2^{31}-1$
无符号整型	unsigned	2字节	0~65535
无符号短整型	unsigned short	2字节	0~65535
无符号长整型	unsigned long	4字节	0~($2^{32}-1$)

运行结果： 32767,-32768

例3.1 整型数据的溢出

```
#include <stdio.h>
void main()
{int a,b;
a=32767;
b=a+1;
printf( "%d,%d\n" ,a,b);
}
```

说明：一个整型变量只能容纳-32768～32767范围内的数。

§3.4 浮点型数据

两种表示形式 { 小数 0.123
 指数 3e-3

注意： e后面的指数必须为整数

§3.4 浮点型数据

一个浮点型数据一般在内存中占4个字节(32位)。

与整型数据的存储方式不同，浮点型数据是按照指数形式存储的(分成小数部分和指数部分，分别存放)。

§3.4 浮点型数据

有单精度（float型）、双精度（double型）和长双精度型（long double）等形式。

类型	位数	数的范围	有效数字
float	32	$10^{-37} \sim 10^{38}$	6~7 位
double型	64	$10^{-307} \sim 10^{308}$	15~16位
long double	128	$10^{-4931} \sim 10^{4932}$	18~19位

§3.4 浮点型数据

C编译系统默认将浮点型常量作双精度来处理。

例如：float f = 2.45678 * 4523.65

先把2.45678和4523.65作为双精度数，进行相乘的运算，得到结果也是双精度数；然后取前7位赋给f。

可在数的后面加字母f或F（如1.65f, 654.87F），编译系统就会把它们按单精度（32位）处理。

§3.5 字符型数据

- (1) 用单引号包含的一个字符是字符型常量
- (2) 只能包含一个字符

例

正确：

'a', 'A', '1'

错误：

'abc'、"a" Y

§3.5字符型数据

以“\”开头的特殊字符称为转义字符

\n	换行
\t	横向跳格
\\"	反斜杠
\b	退格
\r	回车
\ddd	ddd表示1到3位八进制数字
\xhh	hh表示1到2位十六进制数字

§3.5字符型数据

- 字符型变量用来存放字符常量，注意：只能放一个字符。
- 定义形式如下：`char c1,c2;`
可以用下面语句对c1,c2赋值：
`c1= 'a'; c2= 'b' ;`
- 一个字符变量占一个字节。

§3.5 字符型数据

- 实际存储是将该字符的ASCII代码放到存储单元中（见教材附录A）。

这样使字符型数据和整型数据之间可以通用：一个字符数据既可以以字符形式输出，也可以以整数形式输出。还可以进行加减乘除各种运算。

• 运行结果：

a b
97 98

例3.6 向字符变量赋以整数。

```
#include <stdio.h>
void main()
{char c1,c2;
c1=97;
c2=98;
printf( "%c %c\n" ,c1,c2);
printf( "%d %d\n" ,c1,c2);
}
```

- **说明：**在第3和第4行中，将整数97和98分别赋给c1和c2，它的作用相当于以下两个赋值语句：

c1='a'; c2='b';

因为'a'和'b'的ASCII码为97和98

• 运行结果：A B

例3.7 大小写字母的转换

```
#include <stdio.h>
void main()
{char c1,c2 ;
 c1='a';
 c2='b';
c1=c1-32;
c2=c2-32;
printf( "%c %c" , c1,c2) ;
}
```

- **说明：**程序的作用是将两个小写字母a和b转换成大写字母A和B。
- 每一个小写字母比它相应的大写字母的ASCII码大32。
- C允许字符与整数算术运算。

§3.5字符串

- 双撇号括起来的字符序列。

“How do you do.”, “a” ,
“\$123.45”

- 输出方式：

```
printf("How do you do.");
```

插一句：以上只能定义常量。
变量字符串可以借助数组定义。

C规定：在字符串常量的结尾加一个“字符串结束标志”＼0，以便系统判断字符串是否结束。

如：字符串常量“CHINA”在内存中是：

C	H	I	N	A	\0
---	---	---	---	---	----

'＼0'占一个单元内存，但输出时不显示。

§3.7 各类数值型数据间的混合运算

整型、浮点型、字符型可混合运算。

先转换成同一类型，然后运算。

转换由系统自动完成。原则是：

把低精度转为高精度

§3.8 算术运算符和算术表达式

3.8.1 C 运算符简介

附录C汇总了所有运算符优先级、结合方式（单目运算符从右向左、其他从左向右）。

- (1) 算术运算符 ($+$ $-$ $*$ $/$ $\%$)
- (2) 关系运算符 ($>$ $<$ $= =$ $> =$ $< =$ $! =$)
- (3) 逻辑运算符 ($!$ $\&&$ $||$)
- (4) 位运算符 ($<<$ $>>$ \sim $|$ \wedge $\&$)
- (5) 赋值运算符 (= 及其扩展赋值运算符)
- (6) 条件运算符 ($?$ $:$)
- (7) 逗号运算符 (,)

§3.8 算术运算符和算术表达式

- (8) 指针运算符 (*和&)
- (9) 求字节数运算符 (sizeof)
- (10) 强制类型转换运算符 ((类型))
- (11) 分量运算符 (. ->)
- (12) 下标运算符 ([])
- (13) 其他 (如函数调用运算符 ())

§3.8 算术运算符和算术表达式

算术运算符：

- $+$ (加法，或正值)
- $-$ (减法，或负值)
- $*$ (乘法)
- $/$ (除法)
- $\%$ (模，求余均应为整数，如： $7 \% 4$ 的值为 3)。

§3.8 算术运算符和算术表达式

强制类型转换运算符

将一个表达式转换成所需类型。

例如：

- (double) a
- (int)(x+y)
- (float) (5%3)

§3.8 算术运算符和算术表达式

自增、自减运算符

使变量的值增 1 或减 1

如：

- $++i$, $--i$ (在使用 i 之前, 先使 i 的值加 (减) 1)
- $i++$, $i--$ (在使用 i 之后, 使 i 的值加 (减) 1)

§3.8 算术运算符和算术表达式

例如：

① $i=3; j = ++i;$

i 的值先变成4, 再赋给 j , j 的值均为 4

② $i=3; j = i ++;$

先将 i 的值3赋给 j , j 的值为 3 , 然后 i 变为 4

§3.8 算术运算符和算术表达式

注意：避免令人混淆的写法！

例如：i 的初值为 3，如果有下面的函数调用：

printf ("%d , %d" , i , i++)

在有的系统中，从左至右求值，输出“3，3”。
在多数系统中对函数参数的求值顺序是自右而左，
printf 函数输出的是“4，3”。

避免反人类的程序写法！

§3.9 赋值运算符和赋值表达式

赋值运算符 =

如果两侧的类型不一致，在赋值时要进行类型转换。

- ①浮点型数据（包括单、双精度）赋给整型，
舍弃小数部分（不是四舍五入！）。
- ②将整型数据赋给单、双精度，数值不变，
转成浮点数形式存储。

...

§3.9 赋值运算符和赋值表达式

复合的赋值运算符

“=”之前加上其他二目运算符。

例如：

以“ $a += 3$ ”为例来说
明，它相当于使 a 进行一次
自加(3)的操作。即先使
 a 加3，再赋给 a 。

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

§3.9 赋值运算符和赋值表达式

注意：如果右侧包含若干项，则相当于它有括号。

如： $x \% = y + 3$ 相当于

$x = x \% (y + 3)$ (而不是 $x = x \% y + 3$)

C 可以使用 10 种复合赋值运算符：

$+ =$, $- =$, $* =$, $/ =$, $\% =$, $<<=$, $>>=$, $\&=$,
 $\wedge =$, $| =$

所有二元（二目）运
算符都可以

§3.10 逗号运算符和逗号表达式

逗号运算符：

又称为“顺序求值运算符”

如： $3 + 5, 6 + 8$

逗号表达式
的值为14

§3.10 逗号运算符和逗号表达式

一般形式为：

表达式1，表达式2，表达式3，.....，表达式n

它的值为表达式n的值。

逗号运算符的优先级是所有运算符中最低的

例： $x = (a = 3, 6 * 3)$

赋值表达式，
将一个逗号表
达式的值赋给
 x ， x 的值等
于18

§3.10 逗号运算符和逗号表达式

注意：并不是任何地方出现的逗号都是作为逗号运算符。例如函数参数也是用逗号来间隔的。

如： `printf("%d,%d,%d",a,b,c);`

第四章

最简单的C程序设计

§ 4.1 C语句概述（续）

(一) 控制语句 完成一定的控制功能

1 **if()** ~**else** 条件语句

2 **for()**~ 循环语句

3 **while()**~循环语句

4 **do** ~**while();**循环语句

5 **continue** 继续语句

6 **break** 间断语句

7 **switch()** 开关语句

8 **goto** 转向语句

9 **return** 返回语句

§ 4.1 C语句概述（续）

（二）函数调用语句

有一个函数调用加一个分号构成一个语句



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

§ 4.1 C语句概述（续）

(三) 表达式语句

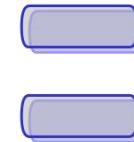
有一个表达式加一个分号构成一个语句

例：

赋值表达式



分号



表达式语句

a = 3 ;

§ 4.1 C语句概述（续）

(四)空语句

只有一个分号的语句

； （什么也不做）

- 用来做流程的转向点
- 用来作为循环语句中的循环体

§ 4.1 C语句概述（续）

(五) 复合语句 用一对{}括起来的语句



```
{      z=x+y;  
      t=z/100;  
      printf("%f",t);  
}
```

§ 4.2 数据的输入输出

C语言本身不提供输入输出语句, 输入和输出由C函数库实现

字符输入输出: getchar putchar

格式输入输出: scanf printf

字符串输入输出: gets puts

§ 4.3 库函数

在使用系统库函数时,要用预编译命“#include” 将有关的“头文件”包括到用户源文件中.

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

或 :

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

· · ·

头文
件

§ 4.4 字符数据的输入输出（续）

例4.1 输出单个字符。

```
#include<stdio.h>
void main()
{
char a,b,c;
a= 'B';b= 'O';c= 'Y';
```

运行结果：B
O
Y

```
putchar(a);putchar('\n');putchar(b);putchar('\n');putchar(c);putchar(d);
```

运行结果：BOY

§ 4.4 字符数据的输入输出（续）

例4.2 输入单个字符。

```
#include<stdio.h>
void main()
{
    char c;
    c=getchar();
    putchar(c);
    putchar( '\n' );
}
```

运行程序：

从键盘输入字符‘a’

按Enter键

屏幕上将显示输出的字符‘a’

a ↴
a

§ 4.5 格式输入与输出

格式输出函数

- 输出若干个任意类型的数据。
- 一般格式：printf（格式控制，输出表列）

%d:带符号的十进制整数

%o:无符号八进制整数

%x:无符号十六进制整数

%u:无符号十进制整数

§ 4.5 格式输入与输出（续）

%c:字符

%s:字符串

%f:小数形式输出单，双精度数，默认输出六位小数

%e:以指数形式输出实数

%g:选用%f或%e格式中输出宽度较短的一种格式，不输出无意义的0

§ 4.5 格式输入与输出（续）

d格式符，输出十进制整数。

%m d : m为某个整数，指定的输出字段的宽度。
如果数据的位数小于m，则左端补以空格；
若大于m，则按实际位数输出。

S 4.5 格式输入与输出（续）

s格式符，输出字符串。

例如：

```
printf ("%s", "CHINA")
```

输出字符串“CHINA”（不包括双引号）。

%ms，输出的字符串占 m 列：

若串长大于m，则全部输出

若串长小于m，则左补空格。

§ 4.5 格式输入与输出（续）

f 格式符，用来以小数形式输出实数（包括单双精度）

① %f。不指定字段宽度，系统自动指定字段宽度。

整数部分全部输出，并输出 6 位小数。

应注意，输出的数字中并非全部数字都是有效数字。

② %m.nf。

指定输出的数据共占 m 列，其中有 n 位小数。

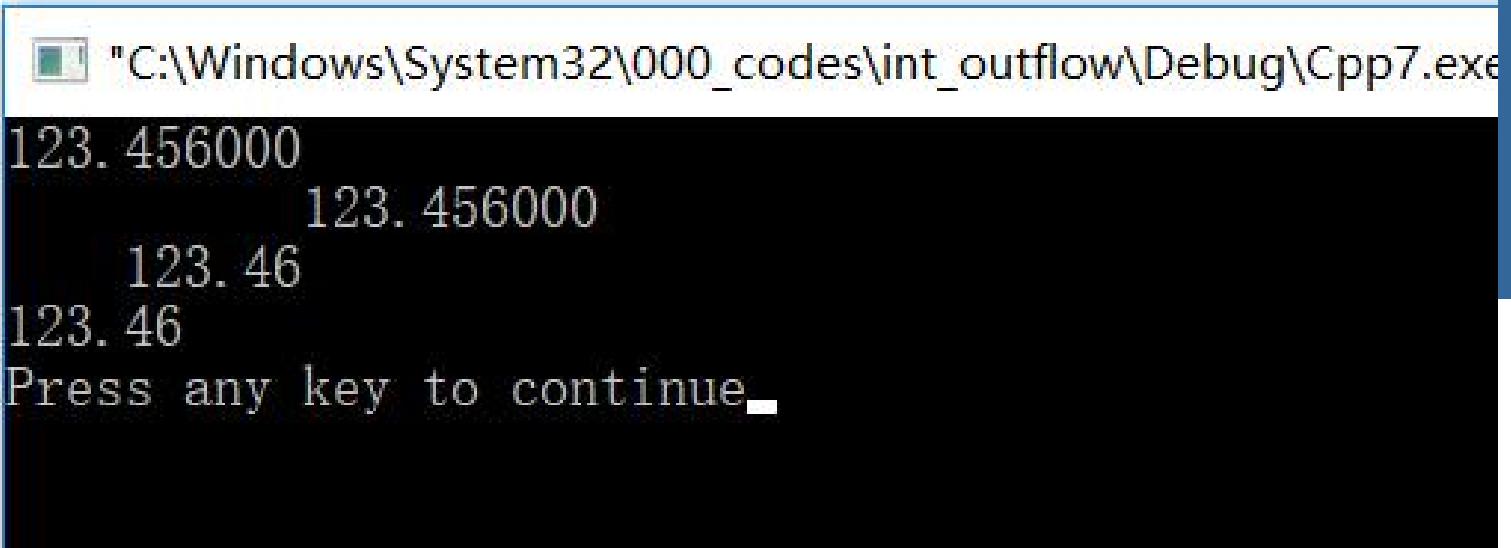
如果数值长度小于 m，则左端补空格。

如果 m 前面加负号，则为右端补空格。

§ 4.5 格式输入与输出（续）

例4.8 输出实数时指定小数位数。

```
#include <stdio.h>
void main()
{
    double f=123.456;
    printf("%f\n%2.0f\n%10.2f\n%.2f\n",f,f,f,f);
}
```



```
"C:\Windows\System32\000_codes\int_outflow\Debug\Cpp7.exe"
123.456000
123.456000
123.46
123.46
Press any key to continue...
```

§ 4.5 格式输入与输出（续）

(8) e 格式符，以指数形式输出实数。

① %e。不指定数据的宽度、小数位数。

例：

```
printf ("%e", 123.456);
```

输出：

1 · 2 3 4 5 6 0	e + 0 0 2
8列	5列

所输出的实数共占 13 列宽度。（注：不同系统的规定略有不同）

② %m.ne 和 %-m.ne。 m、n 和“-”字符的含义与前相同。

§ 4.5 格式输入与输出（续）

格式输入函数

- 按照变量在内存的地址将变量值存进去。
- 一般格式：scanf（格式控制，地址表列）

同printf函数

若干个地址组成的表列

§ 4.5 格式输入与输出（续）

例4.9 用scanf函数输入数据。

```
#include<stdio.h>
void main()
{
int a,b,c;
scanf( "%d%d%d" ,&a,&b,&c );
printf( "%d,%d,%d\n" ,a,b,c );
}
```

a在内存中的地址
&是取地址运算符

运行情况：

3 4 5 ↴
3,4,5

(输入a,b,c的值)
(输出a,b,c的值)

第五章

选择结构程序设计

§5.1 关系运算符和关系表达式

关系运算符

<	(小于)	优先级相同 (高)
1. <=	(小于或等于)	
2. >	(大于)	
3. >=	(大于或等于)	
4. ==	(等于)	
5. !=	(不等于)	优先级相同 (低)

说明：

关系运算符的优先级低于算术运算符

高于赋值运算符

§5.1 关系运算符和关系表达式（续）

关系表达式

- 用关系运算符接起来的式子。

例： $a > b$, $a + b > b + c$, $(a = 3) > (b = 5)$, $'a' < 'b'$, $(a > b) > (b < c)$

- 关系表达式的值是一个逻辑值，即“真”或“假”。

C语言中没有专用的逻辑值，1代表真，0代表假

例：关系表达式” $2 > 1$ ”的值为“真”，表达式的值为1。

§5.2 逻辑运算符和逻辑表达式

逻辑运算符及其优先次序

1. `&&` (逻辑与) 相当于其他语言中的AND
2. `||` (逻辑或) 相当于其他语言中的OR
3. `!` (逻辑非) 相当于其他语言中的NOT

例：`a&&b` 若a,b为真，则`a&&b`为真。

`a||b` 若a,b之一为真，则`a||b`为真。

`! a` 若a为真，则`!a`为假。

优先次序：

任何非零的数值被认作“真”

`!(非)` -> `&&(与)` -> `||(或)`

§5.2 逻辑运算符和逻辑表达式（续）

逻辑表达式的求解中，并不是所有的逻辑运算符都要被执行：

(1) $a \& \& b \& \& c$ 只有a为真时，才需要判断b的值；只有a和b都为真时
才需要判断c的值。

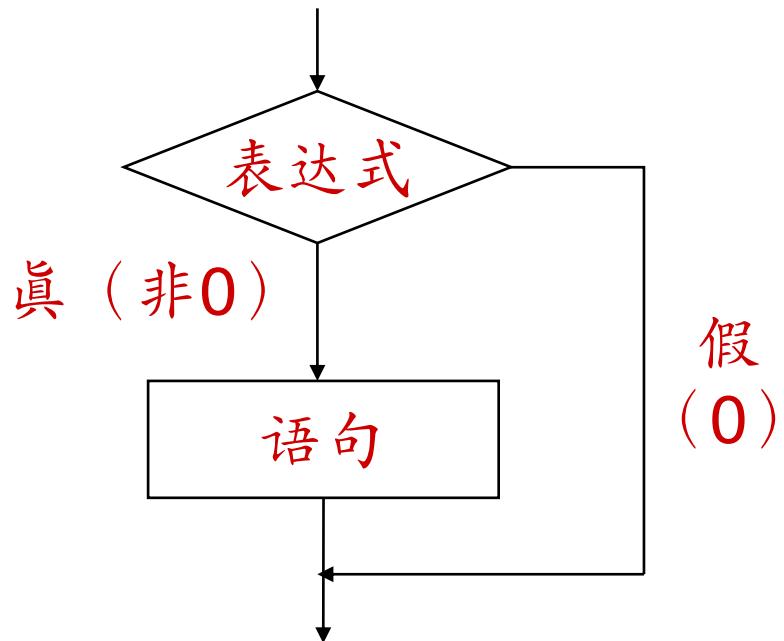
(2) $a || b || c$ 只要a为真，就不必判断b和c的值；只有a为假，才
判断b。a和b都为假才判断c

§5.3 if语句

一、if语句的三种基本形式

(1)

```
if(...)  
{...}
```

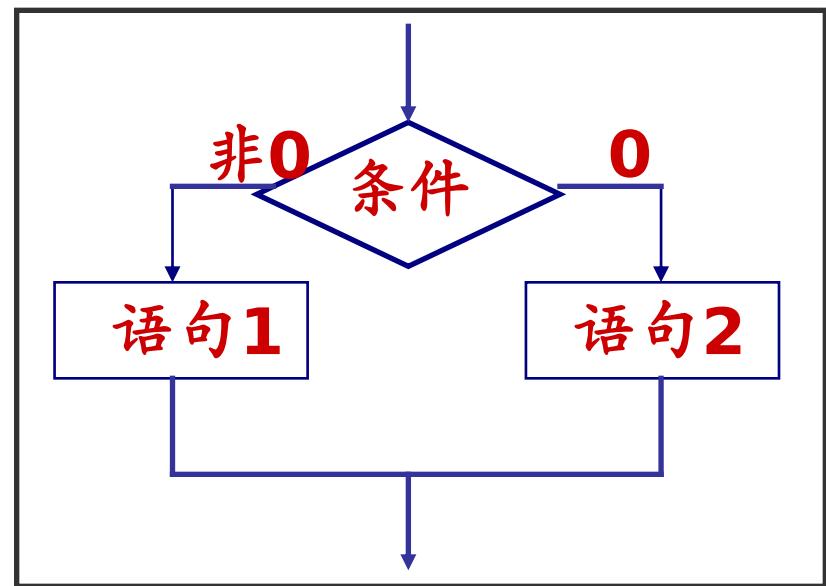


§5.3 if语句（续）

(2)if(...)

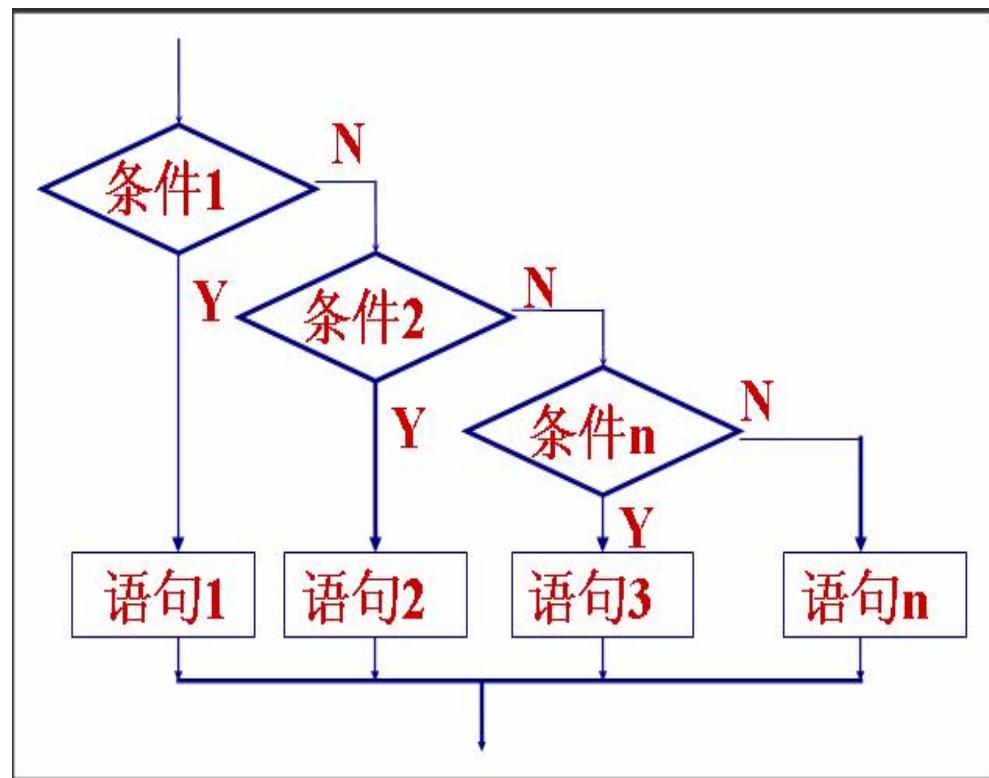
```
...  
else  
...  
...
```

例：if(x>y)
 printf("%d",x);
else
 printf("%d",y);



§5.3 if语句（续）

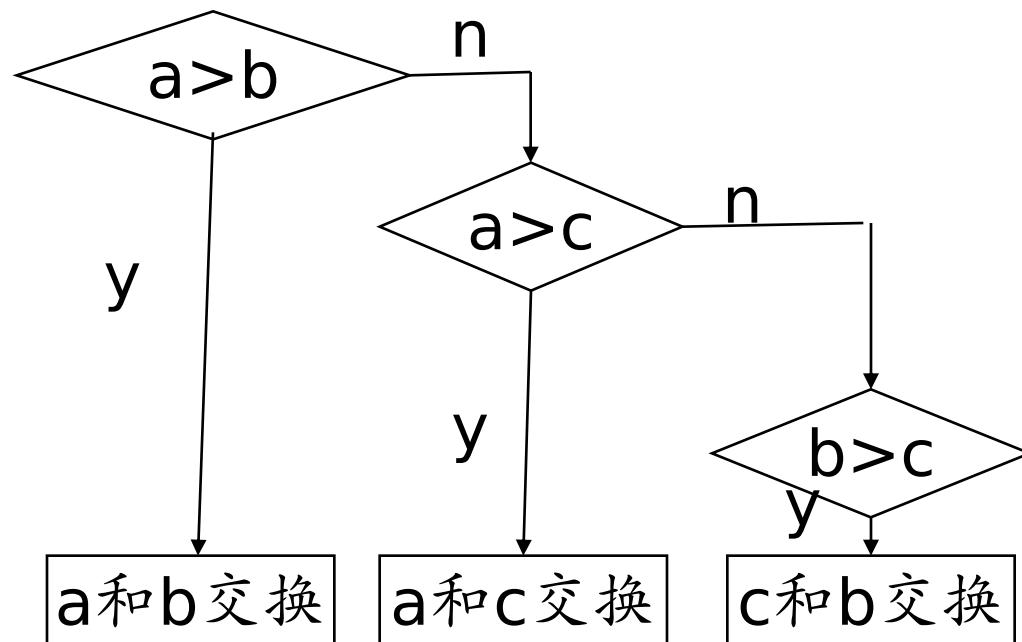
```
(3)if(… ) ...
else if(… ) ...
else if(… ) ...
.....
else if(… ) ...
else ...
```



§5.3 if语句（续）

例5.2 输入三个数a,b,c,按由小到大的顺序输出。

if $a > b$ 将a和b对换
if $a > c$ 将a和c对换
if $b > c$ 将b和c对换



§5.3 if语句（续）

```
#include <stdio.h>
void main ( )
{ float a,b,c,t;
scanf( "%f,%f,%f" ,&a,&b,&c);
if(a>b)
    {t=a;a=b;b=t;}
if(a>c)
    {t=a;a=c;c=t;}
if(b>c)
    {t=b;b=c;c=t;}
printf("%5.2f,%5.2f,%5.2f\n",a,b,c);
}
```

§5.3 if语句（续）

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。

形式：

if()

 if() 语句1
 else 语句2



内嵌if

else

 if() 语句3
 else 语句4



§5.3 if语句（续）

三. 条件运算符

- 格式： 表达式1 ? 表达式2 : 表达式3
- 功能： 判断表达式1的值，如果成立就执行表达式2，否则就执行表达式3

§5.3 if语句（续）

例：

`m a x = (a > b) ? a : b ;`

§5.4 switch语句

switch语句的格式：

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

§5.4 switch语句（续）

例：

要求按照考试成绩的等级输出百分制分数段，用
switch语句实现：

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

§5.4 switch语句（续）

说明：

- (1) `switch`后面括弧内的“表达式”，ANSI标准允许它为任何类型。
- (2) 当表达式的值与某一个`case`后面的常量表达式的值相等时，就从此`case`开始，执行下面所有`case`的语句
- (3) 如遇到`break`语句，就跳出结束`switch`
- (4) 若所有的`case`中的常量表达式的值都没有与表达式的值匹配的，就执行`default`后面的语句。

第六章

循环控制

§6.1 概述

问题1 : $y = \sum_{n=1}^{100} n$

问题2: 求学生平均成绩 分数相加后除以课数

循环结构是结构化程序设计的基本结构。

在许多问题中需要用到循环控制。

§6.2 goto语句以及用goto语句构成循环1

goto不讲了，有兴趣自己看...

- goto语句为无条件转向语句，它的一般形式为

 goto 语句标号；

- 语句标号用标识符表示，它的定名规则与变量名相同，即由字母、数字和下划线组成，其第一个字符必须为字母或下划线。

例如：goto label_1; 合法；

 goto 123； 不合法。

运行结果：5050

例6.1 用if语句和goto语句构成循环，计算1到100的和

```
void main( )
{
    int i, sum=0;
    i=1;
loop:   if(i<=100)
    {
        sum=sum+i;
        i++;
        goto loop;
    }
    printf("%d\n", sum);
```

结构化程序设计方法主张限制使用goto语句，因为滥用goto语句将使程序流程无规律、可读性差。

说明：这里用的是“当型”循环结构，当满足“ $i \leq 100$ ”时执行花括弧内的循环体。

§6.3 用while语句实现循环

while语句用来实现“当型”循环结构。

一般形式：

while (表达式) 语句

当表达式为非0值时，执行while语句中的内嵌语句。其特点是：先判断表达式，后执行语句。

例6.2 求1到100的和

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

运行结果：5050

说明：(1)循环体如果包含一个以上的语句，应该用花括弧括起来，以复合语句形式出现。(2)在循环体中应有使循环趋于结束的语句。

§6.4 用do-while语句实现循环

do-while语句的特点:先执行循环体，然后判断循环条件是否成立。

一般形式: **do**

 循环体语句

while (表达式);

执行过程：先执行一次指定的循环体语句，然后
 循环。

$$\sum_{n=1}^{100} n$$

例6.3 求1到100的和

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

运行结果：5050

§6.4 用do-while语句实现循环

while语句和用do-while语句：

如果while后面的表达式一开始就开始为假(0值)时，两种循环的结果不同的。

§ 6.5 用for语句实现循环

- C语言中的for语句使用最为灵活。
- 一般形式：
for(表达式1；表达式2；表达式3) 语句

§ 6.5 用for语句实现循环

for(表达式1（初值）；表达式2（判断）；
表达式3（语句）) 内嵌语句

顺序：1、2、语句、内嵌语句、3.

(1) 先求1。

(2) 再判断2，若值为真，执行内嵌语句、(3)；
若为假，结束循环。

(3) 转到(2)，继续。。。

§ 6.5 用for语句实现循环

例如：`for(i=1;i<=100;i++) sum=sum+i;`

它相当于以下语句：

`i=1;`

`while(i<=100)`

`{sum=sum+i;i++;}`

显然，用for语句
简单、方便。

§ 6.5 用for语句实现循环

说明：

表达式都可省略，分号不能省略。如：

for(; ;) 语句

相当于

while(1) 语句

无终止地执行循环体。

§ 6.5 用for语句实现循环

说明：

表达式1和表达式3可以用逗号，如：

```
for(sum=0,i=1;i<=100;i++)  
sum=sum+i;
```

§ 6.5 用for语句实现循环

举例：

```
for( ;(c=getchar())!='0';)  
    printf(" %c" , c);
```

f只有表达式2，而无表达式1和表达式3。

作用是每读入一个字符后立即输出该字符，直到输入一个‘0’为止。

运行情况：

Computer 琰 (输入)

Computer (输出)

而不是

CCoommppuutteerr

§ 6.5 用for语句实现循环

注意： C语言中的for语句比其他语言(如BASIC，PASCAL)中的FOR语句功能强得多。

最好不要把与循环控制无关的内容放到for语句中，降低可读性。

§6.6循环的嵌套

- 三种循环(while循环、do-while循环和for循环)可以互相嵌套。

§6.6循环的嵌套

例如：

(1) **while()**

{...

while()

 {...}

}

(2) **do**

{...

do

 {... }

while();
 }

(3) **for(;;)**

{

for(;;)

 {... }

}

(4) **while()**

{...

do{...}

while()

 {...}

}

(5) **for(;;)**

{...

while()

 { }

...

}

(6) **do**

{...

for(;;){ }

...

}

while()

....

§6.7几种循环的比较

- (1)四种循环都可以用来处理同一问题，一般情况下可以互相代替。不提倡goto。
- (2)while循环、do-while循环和for循环，可以用break语句跳出循环，用continue语句结束本次循环。goto语句和if语句构成的循环不行。

§6.8 break语句和continue语句

6.8.1 break语句

提前结束循环，执行循环下面的语句

一般形式：

break;

注意：break语句不能用于循环语句和switch语句之外的任何其他语句中！

§6.8 break语句和continue语句

6.8.2 continue语句

结束本次循环，即跳过循环体中下面尚未执行的语句，接着进行下一次是否执行循环的判定。

一般形式：

continue;

§6.8 break语句和continue语句

continue语句和break语句的区别

continue语句只结束本次循环，而不是像
break终止整个循环的执行。

while(表达式1)

{ ...

if(表达式2) **continue;**

 ...

}

§6.9 程序举例

例6.6 用 $\pi/4 \approx 1 - 1/3 + 1/5 - 1/7 + \dots$ 公式求 π 的近似值，直到某一项的绝对值小于 10^{-6} 为止。

$$\sum_{n=1}^{100} n$$

例6.6 求pi的近似值

```
#include <stdio.h>
#include<math.h>
void main()
{ int s;float n , t , pi ;
    t=1 ; pi=0;n=1.0;s=1;
while(fabs(t)>1e-6)
{pi=pi+t;n=n+2;s=-s;t=s/n;}
    pi=pi*4;
    printf(" pi=%10.6f\n" , pi );
}
```

运行结果：

pi=

3.141594

本次作业（按学号单复数，做一半）：

第三章：

6、7

第四章：

8、9、11、12

第五章：

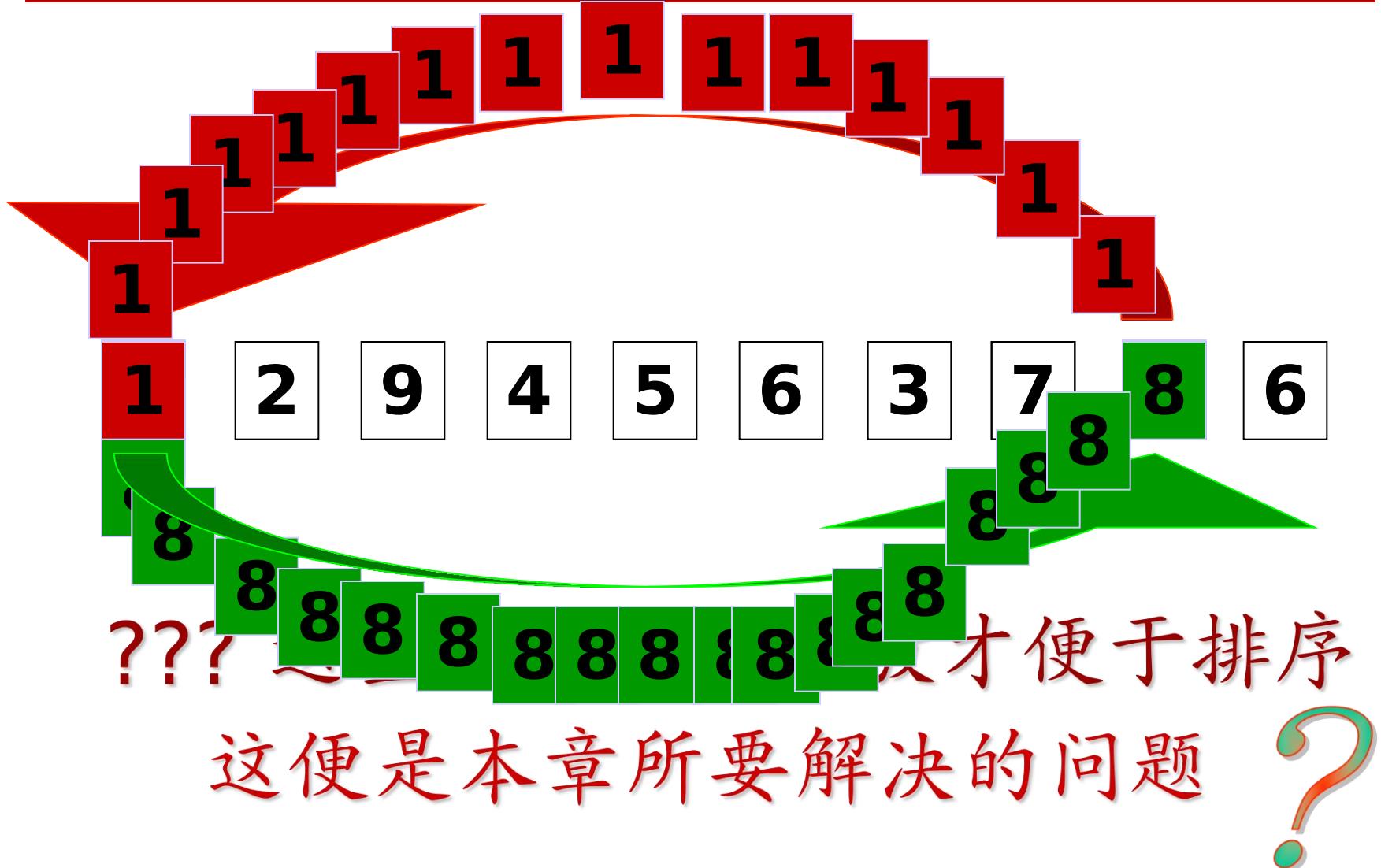
3-17。其中17题所有人都做一下。

第七章

数 组

第七章 数组

问题：给一组数排序，这组数该如何存放



第七章 数组

1 本章要点



一维、二维数组的定义和使用方法



有关数组的算法

§7-1 一维数组的定义和引用

问：如下数据该如何存储呢？

- 一个班学生的学习成绩
- 一行文字
- 一个矩阵

这些数据的特点：

- 1、相同的数据类型；
- 2、使用过程中需要保留原始数据

C语言为这些数据，提供了一种构造数据类型：数组。

数组

一组具有相同数据类型的数据的有序集合。

§ 7.1.1一维数组的定义

1、一维数组的定义格式为：

类型说明符 数组名 [常量表达式]；

例如： int a [10]；

定义了一个整形数组，数组名为a，有10个元素。

2、说明：

(1) 数组名定名规则和变量名相同，遵循标识符定名规则。

(2) 方括弧中的常量表达式指定数组中元素的个数，表示数组长度。例如， $a[10]$ 表示 a 数组有 10 个元素。

(3) 特别注意下标从 0 开始。 $a[10]$ 的 10 个元素是：
 $a[0], a[1], a[2], \dots, a[9]$ 。

不存在数组元素 $a[10]$ 。

(4) 方括弧不能包含变量。也就是说，C 语言不允许对数组的大小作动态定义，数组的大小不依赖于程序运行过程中变量的值。

这样做不允许：

```
int n;  
scanf("%d", &n); /*临时输入数组大小 */  
int a [n] ;
```

其他常见的错误

- ① float a[0]; /* 数组大小为0没有意义 */
- ② int b(2)(3); /* 不能使用圆括号 */
- ③ int k, a[k]; /* 不能用变量说明数组大小*/

3、内存中的存放

一维数组： float mark[100];

每个数据元素占用基
类型的字节数

按指标高低顺序存放

低地址

高地址

86.5	mark[0]
92.0	mark[1]
77.5	mark[2]
52.0	mark[3]
:	:
94.0	mark[99]

§ 7.1.2 数组元素的引用

1、数组元素的引用方式：

数组名 [下标]

下标可以是常量，也可以是变量。例如：

$a[0] = a[5] + a[7] - a[2*3]$

此时[]中的整数代表数组单元的指标，不代表数组长度。

2、一维数组元素引用的程序实例

```
#include <stdio.h>
void main()
{
    int i, a [10];
    for (i=0; i<=9; i++)
        a [i] = i;
    for(i=9; i>=0; i--)
        printf("%d ", a [i]);
    printf("\n");
}
```

运行结果如下：

9 8 7 6 5 4 3 2 1 0

程序使a [0] 到
a [9] 的值为0~9，
然后按逆序输出。

§ 7.1.3 数组的初始化

实现方法：

(1) 定义数组时，对数组元素赋初值。如：

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

将初值依次放在一对花括弧内。

2) 可以只给一部分元素赋值。例如：

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

定义a数组有10个元素，但花括弧内只提供5个初值，
这表示只给前面5个元素赋初值。

(3) 如果想使一个数组中全部元素值为0，

可以写成

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

不能写成：int a [10] ={0*10};

这是与FORTRAN语言不同的，不能给数组整体赋初值。

4) 赋初值时可以不指定数组长度：

例如：`int a [5] ={1, 2, 3, 4, 5};`

也可以写成 `int a [] = {1, 2, 3, 4, 5};`

在第二种写法中，花括弧中有5个数，系统就会据此自动定义a数组的长度为5。

但若数组长度与提供初值的个数不相同，长度不能省略（如，想定义长度为10，就必须写成 `int a [10] ={1, 2, 3, 4, 5};`）。

程序举例1：用数组来处理,求解**Fibonacci**数列

Fibonacci数列公式：已知： $a_1=a_2=1$ $a_n=a_{n-1}+a_{n-2}$
即：1,1,2,3,5,8,13

程序实例：

```
#include <stdio.h>
void main()
{
    int i;
    int f [20] ={1 , 1};
```

```

for(i=2;i<20;i++)
f[i]=f[i-2]+f[i-1];
for(i=0;i<20;i++)
{
    if(i%5==0) printf("\n");
    printf(" %12d" , f[i])
} /*For循环结束*/

```

if语句用来控制换行，每行输出5个数据。

运行结果如下：

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

§ 7.1.4 程序举例

程序举例：用起泡法对10个数排序(由小到大)。

起泡法的思路是：将相邻两个数比较，将小的调到前头。

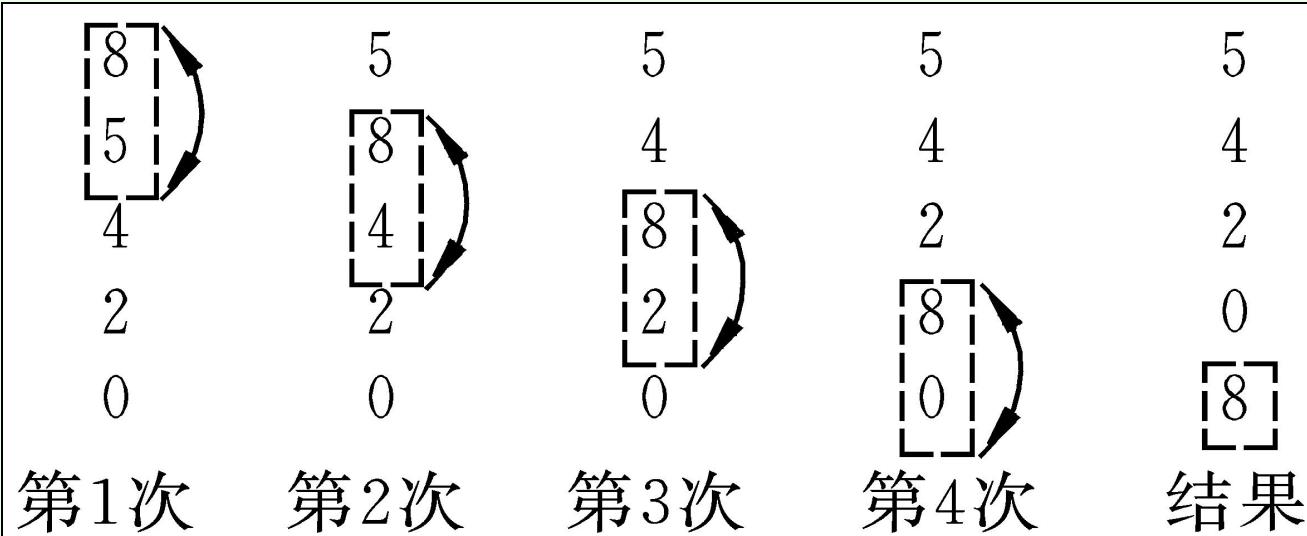
第一趟比较

第1次	第2次	第3次	第4次	第5次	结果
9 8 5 4 2 0	8 9 5 4 2 0	8 5 9 4 2 0	8 5 4 9 2 0	8 5 4 2 9 0	8 5 4 2 0 [9]

注

经过第一趟(共5次比较与交换)后，最大的数9已“沉底”。然后进行对余下的前面5个数第二趟比较，

第二趟比较



注

经过第二趟(共4次比较与交换)后，得到次大的数8。

如果有 n 个数，则要进行 $n-1$ 趟比较。

在第 j 趟，进行前 $n-j$ 个元素的两两比较。

核心代码如下：

```
int a[10], i, j, ;  
...  
for(j=0; j<9; j++)  
    for(i=0; i<9-j; i++)  
        if (a [ i ] >a [ i+1 ] )  
        {  
            t=a[ i ];a[ i ]=a[ i+1 ];a [ i+1 ] =t;  
        }
```

§7-2 二维数组

§ 7.2.1 二维数组的定义

一般形式为

类型说明符 数组名 [常量表达式] [常量表达式] ;

例如：

float a [3] [4] , b [5] [10] ;

不能写成 float a [3, 4] , b [5, 10] ; 

引用方式类似。如：

a [0] [0] = 12.5;

注意

可以把二维数组看作一种特殊的一维数组，其每个都是一个一维数组。

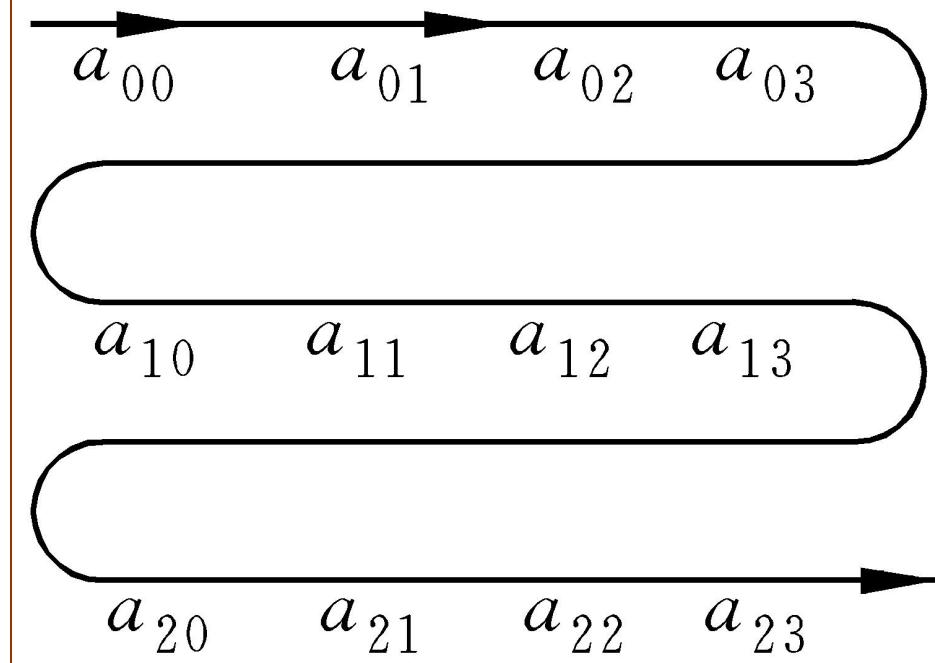
a	[0]	-----	a_{00}	a_{01}	a_{02}	a_{03}
	[1]	-----	a_{10}	a_{11}	a_{12}	a_{13}
	[2]	-----	a_{20}	a_{21}	a_{22}	a_{23}

在内存中的存放

排列顺序是按行存放。

先顺序存放第一行的元素，再存放第二行的元素...

a [3] [4] 数组存放顺序



更高维数组例此可知：

定义三维数组： float a [2] [3] [4] ;

存放顺序如下。指标越高，变化越慢。

三维数组的元素排列顺序

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

在使用数组元素时，应注意下标值应在已定义的数组大小的范围内。否则会引用到存储范围以外的内存单元，程序不报错，运行结果数值奇怪、修改内存产生问题。

常出现的错误：

```
int a [3] [4]; /* 定义a为3×4的数组 */
```

```
|
```

```
a [3] [4] =3; /* 指标溢出，修改了范围外内存，能造成危险后果 */
```

可以用4种方法对二维数组初始化

(1) 分行赋初值。如：

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

(2) 写在一个花括弧内，按排列的顺序赋初值：

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

(3) 对部分元素赋初值。如

```
int a [3] [4] ={{1}, {5}, {9}};
```

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

(4)全部元素赋初值，第一维的长度不指定，第二维的
长度不能省：

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

经测试 int a [3] [] ={1, 2, 3, 4, 5, 6, 7, 8,
9, 10, 11, 12}会报错。编译系统小学数学有待提高。

§7-3 字符数组

定义方法类似。例如：

```
char c[10]={ 'I', ' ', 'a', 'm', ' ', 'h', 'a',
'p', 'p', 'y'};
```

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

§ 7.3.2 字符数组的初始化

对字符数组初始化，最容易理解的方式是逐个字符赋给数组中各元素。如：

- 如果在定义字符数组时不进行初始化，则数组中各元素的值是不可预料的。
- 如果花括弧中提供的初值个数(即字符个数)大于数组长度，则按语法错误处理。

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

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

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

§ 7.3.4 字符串和字符串结束标志

可以用字符串常量来使字符数组初始化。

注意

```
char c [] = "I am happy";
```

```
char c [100] = "I am happy";
```

需要说明的是：字符数组并不要求包含'\\0'。如：

```
char c [5] = {'C', 'h', 'i', 'n', 'a'};
```

这样写是合法的。但是人们为了处理方法一致，在字符数组中也常人为地加上一个'\\0'。如：

```
char c [6] = {'C', 'h', 'i', 'n', 'a', '\\0'}; \
```

§7-3 字符数组

§ 7.3.5 字符数组的输入输出

有两种方法：

- 用格式符 “%c”逐个输入输出。
- 用 “%s”格式符将整个字符串一次输入或输出。

例如

```
char c [ ] = { " China" };  
printf( " %s" , c);
```

在内存中数组c的状态

c 数组

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

说明

- (1) 如果数组长度大于字符串实际长度，也只输出到遇 '\0' 结束。
- (2) 输出字符不包括结束符 '\0'。
- (3) 如果包含一个以上 '\0'，则遇第一个 '\0' 时输出就结束。
- (4) 可以用 scanf 函数输入一个字符串。

例如 `scanf(" %s" , c);`

scanf 函数中的输入项 c 是已定义的字符数组名，输入的字符串应短于已定义的字符数组的长度。

但 scanf 难以处理空格。

例：

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

如果输入：

How are you? 璞

结果会是什么呢？

系统会把空格字符作为字符串之间的分隔符，因此系统只会将 “How”送到str中，后面不传递。输出：

How

§ 7.3.6字符串处理函数

1. puts (字符数组)

- 将一个字符串(以'\\0'结束的字符序列)输出到终端。
- 可以用printf函数输出字符串，因此用的不多。

§ 7.3.6字符串处理函数

2. gets(字符数组)

从终端输入一个字符串到字符数组（函数值是字符数组的起始地址，用得不多）。

puts和gets函数只能输入或输出一个字符串，不能写成 `puts(str1, str2);`
`gets(str1, str2)`

§ 7.3.6字符串处理函数

3. `strcat(字符数组1, 字符数组2)`

连接两个字符数组中的字符串，把字符串2接到字符串1的后面，结果放在字符数组1中。函数值为字符数组1的地址。

§ 7.3.6字符串处理函数

4. strcpy(字符数组1，字符串2)

将字符串2复制到字符数组1中去。

1. 字符数组1的长度不应小于字符串2的长度。
2. 字符数组1必须写成数组名形式(如str1)，字符串2可以是字符数组，也可以是一个字符串常量。

如strcpy(str1, " China");

3. 复制时连'\\0'一起复制。
4. 可以只将字符串2若干个字符复制到数组1。如：

strcpy(str1, str2, 2);

只复制前两个字符，然后加'\\0'。

§ 7.3.6字符串处理函数

5. strcmp(字符串1，字符串2)

比较字符串1和字符串2。

例如：strcmp(str1, str2);

```
strcmp(" China" , " Korea" );
```

```
strcmp(str1, " Beijing" );
```

规则与其他语言相同：从左至右逐个按ASCII码值比较，直到出现不同或遇到'\\0'。

如全部字符相同，则认为相等；若出现不相同的字符，则以第一个不相同的字符的比较结果为准。

- (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，函数值为一负整数。

注意！字符串数组比较，必须使用strcmp：

```
if(strcmp(str1, str2)>0) printf(" yes");
```

不能使用如下形式（貌似会比较数组地址前后）：

```
if(str1>str2) printf(" yes");
```

§ 7.3.6字符串处理函数

6. **strlen** (字符数组)

得到字符串字符数目，不包括'\\0'在内。

如：`char str [10] ={" China" };`
`printf(" %d" , strlen(str));`

输出结果5。

也可以用于字符串常量，如
`strlen(" China");`

§ 7.3.6字符串处理函数

7. **strlwr** (字符串)

将字符串中大写字母换成小写字母。

8. **strupr** (字符串)

将字符串中小写字母换成大写字母。

以上介绍了常用的8种字符串处理函数，注意：
不同的编译系统提供的函数数量、函数名、函数功能不尽相同，使用时要小心。

作业：第六章课后习题1-6,8-9，按学号单复数只做一半。

加一道补充题，所有人都做：：

最简单的计算器，1位正整数的四则运算。

要求用gets()输入字符串，格式如：

1+2-3*4/5-6

用字符数组存放上述表达式，再计算该表达式结果。

要求具有纠错功能，即输入字符不能为数字、+-*/之外所有字符，且所有的数字只有1位。

如做不出来，可考虑只有+-，没有*/的简化情形。

再次强调：程序有适当注释，格式美观，非常重要！

形成稳扎稳打、模块鲜明的节奏！这样才能写大程序。

下面展示两个例子：

某大牛写的 C++ 星系统计程序

<https://github.com/damonge/CUTE/blob/master/CUTE/src/correlator.c>

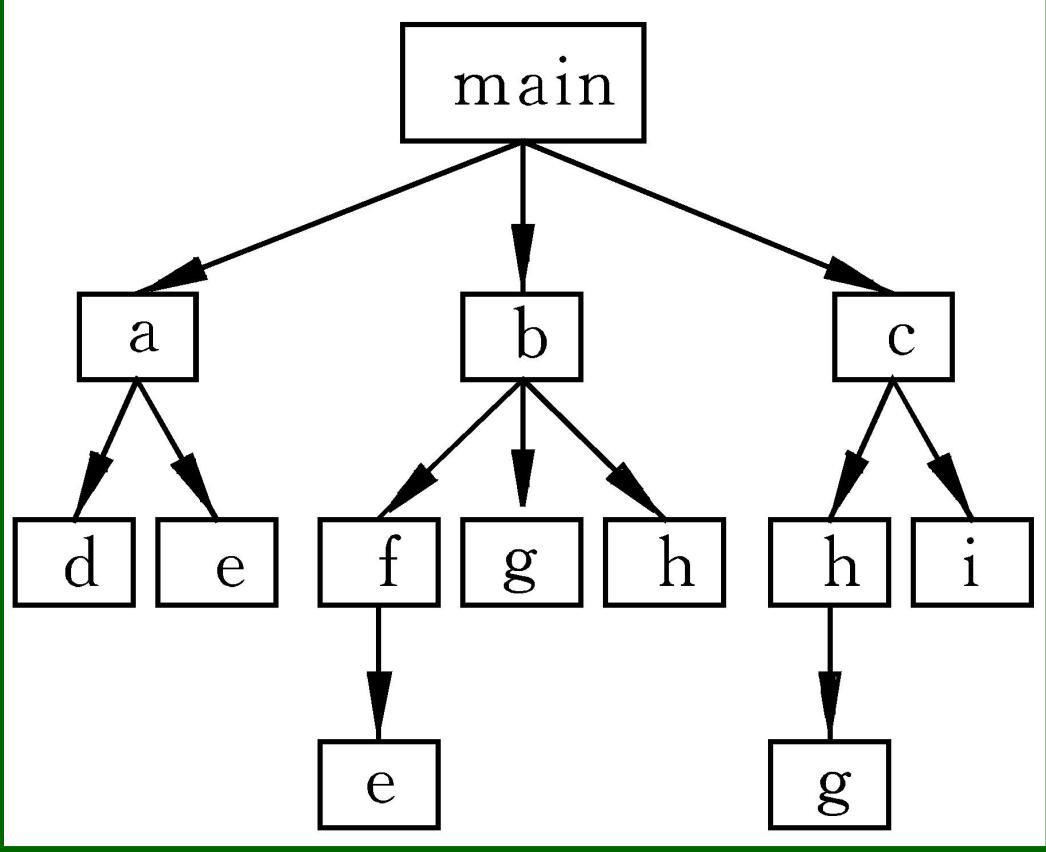
我写的 Fortran 宇宙学程序

<https://github.com/xiaodongli1986/delib>

第八章

函数

§8.1概述



一个较大的程序可分为若干个程序模块，每一个模块用来实现一个特定的功能。
在高级语言中用子程序实现模块的功能。子程序由函数来完成。
一个C程序可由一个主函数和若干个其他函数构成。

由主函数调用其他函数，其他函数也可以互相调用。
同一个函数可以被一个或多个函数调用任意多次。

例8.1先举一个函数调用的简单例子

```
# include <stdio.h>
void main()
{
    void printstar();          /*对printstar函数声明*/
    void print_message();      /*对print_message函数声明
*/
    printstar();               /*调用printstar函数*/
    print_message();           /*调用print_message函数*/
    printstar();               /*调用printstar函数*/
}
```

```
void printstar() /*定义printstar函数*/  
{  
    printf("* * * * * * * * * *\n");  
}
```

```
void print_message() /*定义print_message函数*/  
{  
    printf("How do you do!\n");  
}
```

运行情况如下：

* * * * * * * * * * * * *

How do you do!

* * * * * * * * * * * * *

说明：

(1) 对较大的程序，一般不希望把所有内容全放在一个文件中，而是将他们分别放在若干个源文件中，再由若干源程序文件组成一个C程序。

这样便于分别编写、分别编译，提高调试效率。

一个源程序文件可以为多个C程序公用。

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

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

(4) 所有函数都是平行的，即在定义函数时是分别进行的，是互相独立的。一个函数并不从属于另一函数，即函数不能嵌套定义。函数间可以互相调用，但不能调用main函数。main函数是系统调用的。

(5) 从用户使用的角度看，函数有两种：

- ① 标准函数，即库函数。这是由系统提供的，用户不必自己定义这些函数，可以直接使用它们。不同的C系统提供的库函数的数量和功能基本差不多，但也会有一些不同。
- ② 用户自己定义的函数。用以解决用户的专门需要。

(6) 从函数的形式看，函数分两类：

- ① 无参函数。如例8.1中的printstar和print_message就是无参函数。在调用无参函数时，主调函数不向被调用函数传递数据。无参函数一般用来执行指定的一组操作。例如，例8·1程序中的printstar函数。
- ② 有参函数。在调用函数时，主调函数通过参数向被调用函数传递数据，一般情况下，执行被调用函数时会得到一个函数值，供主调函数使用。

§8.2 函数定义的一般形式

§8.2.1. 无参函数的定义一般形式

定义无参函数的一般形式为：

类型标识符 函数名 ()

{

声明部分

语句部分

}

在定义函数时要用“类型标识符”指定函数值的类型，即函数带回来的值的类型。

例8.1 中的
printstar和
print_message函
数为void类型，
表示不需要带回
函数值。

§8.2.2. 有参函数定义的一般形式

定义有参函数的一般形式为：

类型标识符 函数名 (形式参数表列)

{

声明部分

语句部分

}

例如：

```
int max (int x, int y)
{ int z; /* 函数体中的声明部分 */
    z = x > y ? x : y;
    return (z);
}
```

§8.3 函数参数和函数的值

§8.3.1 形式参数和实际参数

在有参函数中，函数名后面括弧中的变量名称为“形式参数”（简称“**形参**”）。

在主调函数中调用一个函数时，函数名后面括弧中的参数（可以是一个表达式）称为“实际参数”（简称“**实参**”）。

return后面的括弧中的值()作为函数带回的值（称**函数返回值**）。

例8.2 调用函数时的数据传递

```
#include <stdio.h>

void main ()
{ int max(int x, int y);
  /* 对max函数的声明 */
  int a, b, c;
  scanf ("%d,%d", &a, &b);
  c=max (a, b); /* 此处ab是实参*/
  printf ("Max is %d", c);
}
```

```
int max(int x , int y )/*定义有参函数max */  
/* 此处 x y 是形参*/  
{  
    int z ;  
    z = x > y ? x : y ;  
    return ( z ) ;  
}
```

运行情况如下：

7 , 8 ↘ 疑

Max i s 8

通过函数调用，使两个函数中的数据发生联系

```
c=max(a,b);      (main 函数)
```

```
int max(int x,int y) (max 函数)
```

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

在不同的函数之间传递数据，可以使用的方法：

- ◆ 参数：通过形式参数和实际参数
- ◆ 返回值：用 return 语句返回计算结果
- ◆ 全局变量：外部变量

形参和实参的区别

英文名字不同：形参是parameter，实参是argument。

本质不同：形参的本质是一个名字，不占用内存空间。实参的本质是一个变量，已经占用内存空间。

在调试的时候，parameter就转变成argument，这时也往往不使用argument一词，而是称之为variable（变量），因为实参本质上就是一个变量，在内存中占用一块空间。

通过函数调用，使两个函数中的数据发生联系

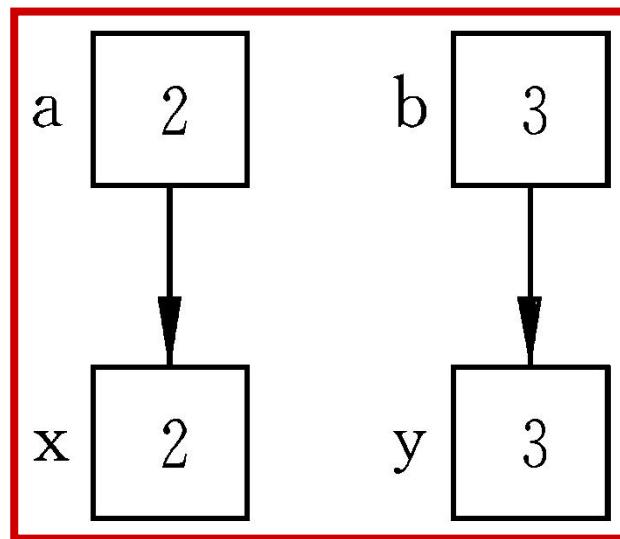
```
c=max(a,b);      (main 函数)
```

```
int max(int x,int y) (max 函数)
```

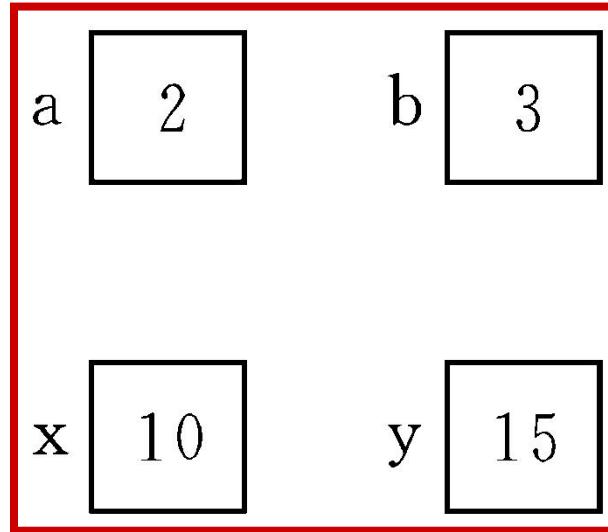
```
{int    z;  
z=x>y?  x:  y;  
return(z);}
```

定义函数中指定的形参，在未出现函数调用时，它们并不占内存中的存储单元。只有在发生函数调用时，函数max中的形参才被分配内存单元。在调用结束后，形参所占的内存单元也被释放。

在C语言中，实参向对形参的数据传递是“值传递”，单向传递，只由实参传给形参，而不能由形参传回来给实参。在内存中，实参单元与形参单元是不同的单元。



在调用函数时，将实参对应的值传递给形参，调用结束后，形参单元被释放，实参单元仍保留并维持原值。因此，在执行一个被调用函数时，形参的值如果发生改变，并不会改变主调函数的实参的值。例如，若在执行函数过程中 x 和 y 的值变为 10 和 15，而 a 和 b 仍为 2 和 3。



§8.3.2 函数的返回值

通常，希望通过函数调用使主调函数能得到一个确定的值，这就是**函数的返回值**。例如，例8.2中，`max(2, 3)` 的值是3。

关于函数返回值的一些说明：

(1) 函数的返回值是通过函数中的**return**语句获得的。

如果需要从被调用函数带回一个函数值供主调函数使用，被调用函数中必须包含return语句。如果不需
要，可以不要return语句。

一个函数中可以有一个以上的return语句，执行到哪
一个return语句，哪一个语句起作用。return语句后
面的括弧也可以不要，

如：“**return z;**” 等价于 “**return (z);**”

(2) 对于不带回值的函数，应当用“void”定义
函数为“无类型”（或称“空类型”）。这样，系
统就禁止在调用函数中使用被调用函数的返回
值。此时在函数体中不得出现return语句。

例 8.3 返回值类型与函

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

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

运行情况如下：

1 . 5 , 2 . 5 ✓ 琢
Max is 2

§8.4 函数的调用

§8.4.1 函数调用的一般形式

函数调用的一般形式为： 函数名（实参表列）

如果是调用无参函数，则“实参表列”
可以没有，但括弧不能省略。

§8.4.2 函数调用的方式

按函数在程序中出现的位置来分，可以有以下三种函数调用方式：

1 · 函数语句

如例8.1中的printstar()。

2 · 函数表达式

函数出现在一个表达式中，例如：

c = 2 *max (a , b) ;

3 · 函数参数

函数作为一个函数的实参。例如：

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

§8.4.3 对被调用函数的声明和函数原型

在一个函数中调用另一函数（即被调用函数）需要具备哪些条件呢？

(1) 首先被调用的函数必须是已经存在的
函数（是库函数或用户自己定义的函数）。
但光有这一条件还不够。

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

(3) 如果使用用户自己定义的函数，而该函数的位置在主调函数的后面（在同一个文件中），则应该在主调函数中或之前，使用函数原型对被调用的函数作声明。

函数原型的一般形式为

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

“声明”一词的原文是declaration，**作用**是把函数名、函数参数的个数和参数类型等信息通知编译系统，以便在遇到函数调用时，编译系统能正确识别函数并检查调用是否合法。
(例如，函数名是否正确，实参与形参的类型和个数是否一致)。

注意：函数的“定义”和“声明”不是一回事。

函数的定义是指对函数功能的确立。

而函数的声明的作用则是把函数的名字、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。

例8·5 对被调用的函数作声明

```
# include <stdio.h>
void main ()
{
    float add (float x, float y) ;
        /*对被调用函数add的声明*/
    float a, b, c ;
    scanf ("%f,%f" , &a , &b) ;
    c=add (a, b) ;
    printf (" sum is %f \n" , c) ;
}
float add (float x, float y)          /*函数首部*/
{
    float z ;                      /* 函数体 */
    z=x+y ;
    return (z) ;
}
```

如果 被调用函数的定义出现在主调函数之前，可以不必加以声明。因为编译系统已经先知道了已定义函数的有关情况，会根据函数首部提供的信息对函数的调用作正确性检查。

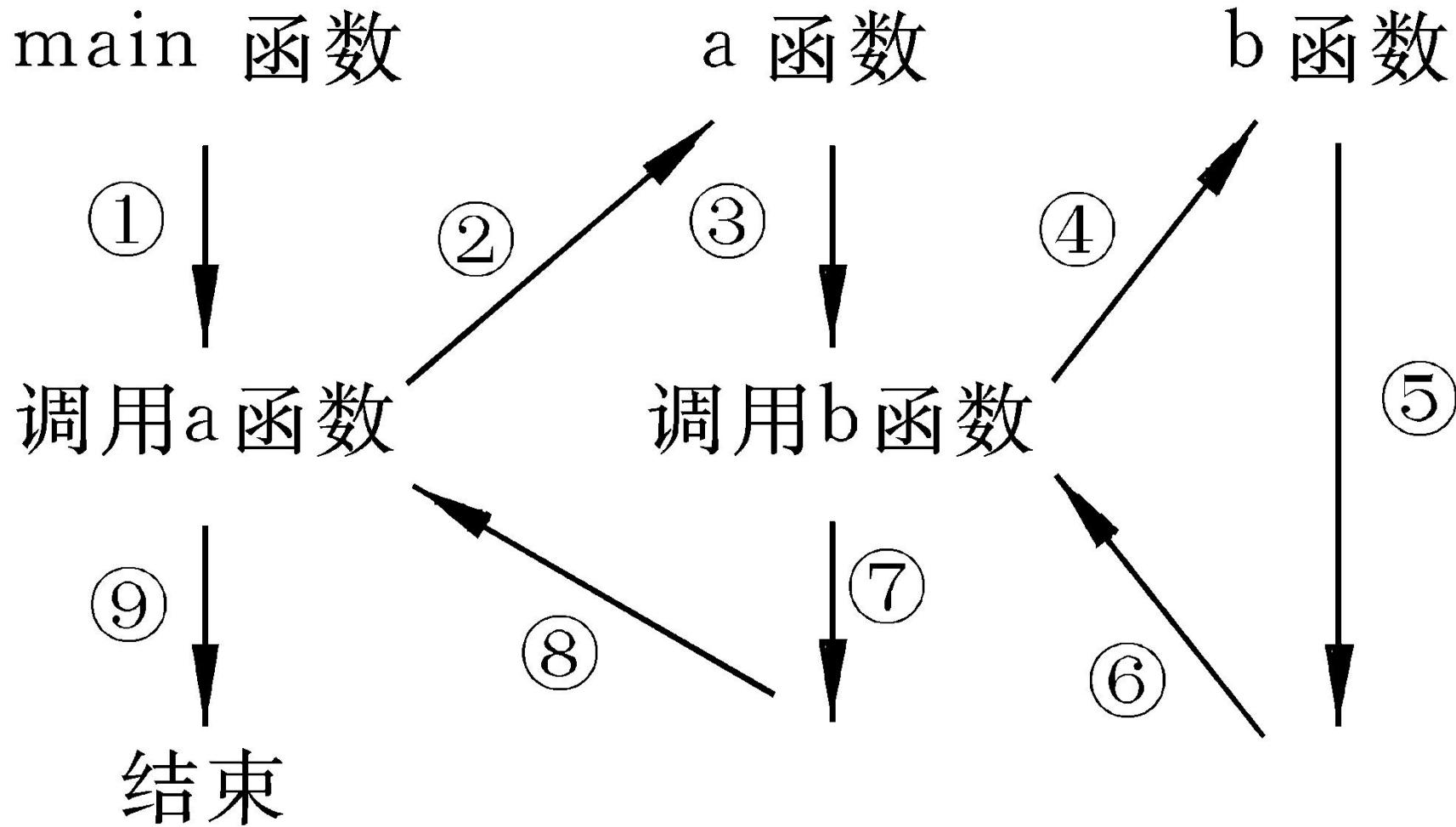
改写例 8.

```
# include <stdio.h>
float add (float x , float y ) /*函数首
部*/
{
    float z ; /* 函数体 */
    z=x+y ;
    return (z) ;
}
void main ()
{
    float a , b , c ;
    scanf ("%f , %f" , &a , &b) ;
    c=add (a , b) ;
    printf (" sum is %f \n" , c) ;
}
```

§8.5 函数的嵌套调用

嵌套定义就是在定义一个函数时，其函数体内又包含另一个函数的完整定义。

C语言不能嵌套定义函数，但可以嵌套调用函数，也就是说，在调用一个函数的过程中，又调用另一个函数。



§8 · 6 函数的递归调用

在调用一个函数的过程中又出现直接或间接地调用该函数本身，称为函数的递归调用。C 语言的特点之一就在于允许函数的递归调用。

例 8·7 有5个人坐在一起，问第5个人多少岁？
他说比第4个人大2岁。问第4个人岁数，他说比第
3个人大2岁。问第3个人，又说比第2个人大2岁
问第2个人，说比第1个人大2岁。最后问第1个
人，他说是10岁。请问第5个人多大。

可以用数学公式表述如下：

$$\text{age}(n) = 10 \quad (n = 1)$$

$$\text{age}(n - 1) + 2 \quad (n > 1)$$

可以用一个函数来描述上述递归过程：

```
int age (int n)          /*求年龄的递归函数*/  
{ int c; /* c用作存放函数的返回值的变量 */  
    if (n == 1) c = 10;  
    else c = age (n - 1) + 2;  
    return (c);  
}
```

用一个主函数调用age函数，求得第5人的年龄。

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

```
void main ()  
{  
    printf ("%d", age (5))  
}
```

运行结果如下：
18

例 8.8 用递归方法求 n !

求 $n!$ 也可以用递归方法，即 $5! = 4! \times 5$ ，而 $4! = 3! \times 4 \dots 1! = 1$ 。可用下面的递归公式表示：

$$n! = 1 \quad (n = 0, 1)$$

$$n \cdot (n - 1)! \quad (n > 1)$$

§8.7 数组作为函数参数

可以用数组名作函数参数，此时形参应当用数组名或用指针变量。

例8.11 有一个一维数组 score，内放10个学生成绩，求平均成绩。

例 8.1 2形参数组可以不定义长度

```
float average (float array [ ] , int  
n )  
{ int i ;  
    float aver , sum=array [ 0 ] ;  
    for ( i = 1 ; i < n ; i ++ =  
        sum=sum+array [ i ] ;  
    aver=sum/n ;  
    return ( aver ) ;  
}
```

调用语句省略。

例 8.13 用选择法对数组中10个整数按由小到大排序。所谓选择法就是先将10个数中最小的数与a〔0〕对换；再将a〔1〕到a〔9〕中最小的数与a〔1〕对换.....每比较一轮，找出一个未经排序的数中最小的一个。共比较9轮。

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

§8.8局部变量和全局变量

§8.8.1局部变量

在一个函数内部定义的变量是内部变量，它只在本函数范围内有效，在此函数以外是不能使用这些变量的。这称为“局部变量”。

不同函数中可以使用相同名字的变量，它们代表不同的对象，互不干扰。

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

```
void main ( )
{int a,b;
...
{int c;
c=a+b; c在此范围内有效 a,b在此范围内有效
...
}
...
}
```

§8.8.2 全局变量

在函数之外定义的变量称为**外部变量**,外部变量是全局变量(也称全程变量)。

全局变量可以为本文件中其他函数所共用。它的有效范围为从定义变量的位置开始到本源文件结束。

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

全局变量p,q的作用范围
全局变量c1,c2的作用范围

全局变量有时非常有用，但不在必要时，也不要滥用。原因如下：

1. 使用全局变量过多，会降低程序的清晰性，在各个函数执行时都可能改变外部变量的值，程序容易出错。
2. 它使函数的通用性降低了，因为函数在执行时要依赖于其所在的外部变量。如果将一个函数移到另一个文件中，还要将有关的外部变量一起移过去。
3. 若外部变量与其他文件的内部变量同名时（此时内部变量起作用），容易出现混淆，降低程序的可靠性和通用性。

一般要求把C程序中的函数做成一个封闭体，除了可以通过“实参——形参”的渠道与外界发生联系外，没有其他渠道。

§8.9 变量的存储类别

§8.9.1 动态存储方式与静态存储方式

从变量的作用域（即从空间）角度来分，可以分为全局变量和局部变量。

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

所谓静态存储方式是指在程序运行期间由系统分配固定的存储空间的方式。而动态存储方式则是在程序运行期间根据需要进行动态的分配存储空间的方式。

数据在内存中存储的方式,分为两大类：
静态存储类和动态存储类。

具体包含四种：

自动的（`auto`），静态的（`static`），
寄存器的（`register`），外部的（`extern`）。

根据变量的存储类别，可以知道变量的作用域和生存期。

§8.9.2 auto变量

函数中的局部变量，如不专门声明为static存储类别，都是动态地分配存储空间的，数据存储在系统分配的动态存储区中。

在调用该函数时系统会给它们分配存储空间，在函数调用结束时就自动释放这些存储空间。因此这类局部变量称为自动变量。

自动变量用关键字auto作存储类别的声明。例如：

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

8.9.3用static声明局部变量

有时希望函数中的局部变量的值在函数调用结束后不消失而保留原值，即其占用的存储单元不释放，在下一次该函数调用时，该变量已有值，就是上一次函数调用结束时的值。

这时就应该指定该局部变量为“静态局部变量”，用关键字 static 进行声明。

通过下面简单的例子可以了解它的特点。

例8·18 输出1到5的阶乘值。

```
#include <stdio.h>
void main ()
{ int fac (int n) ;
    int i ;
    for ( i = 1 ; i <= 5 ; i ++ )
        printf ("%d !=%d\n", i ,fac ( i ) ) ;
}
Int fac (int n)
{ static int f = 1 ; /*在第一次调用时，初值为1 */
*f =
    f = f *n ; /*以后每次调用f的值都会改变 */
    return ( f ) ;
}
```

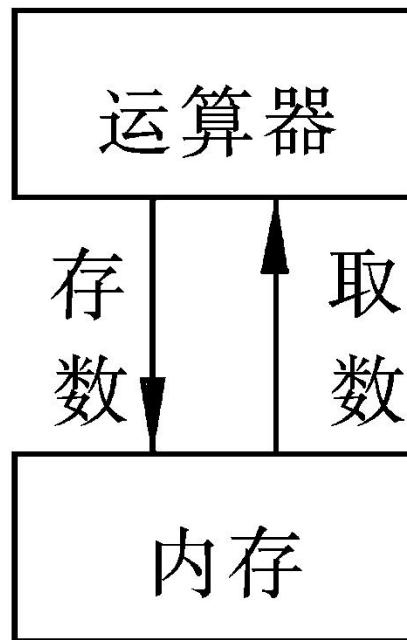
对静态局部变量的说明：

- (1) 在静态存储区内分配存储单元。在程序整个运行期间都不释放。
- (2) 静态变量是在编译时赋初值的，即只赋初值一次，程序运行时它已有初值。以后每次调用函数时不再重新赋初值而只是保留上次函数调用结束时的值。
- (3) 如在定义静态局部变量时不赋初值的话，编译时自动赋初值0（对数值型变量）或空字符（对字符变量）。而对自动变量来说，如果不赋初值则它的值是一个不确定的值；这是由于每次调用时所分配的单元中的值是不确定的。
- (4) 虽然静态局部变量在函数调用结束后仍然存在，但其他函数是不能引用它的。

8.9.4 register变量

一般情况下，变量（包括静态、动态存储方式）的值是存放在内存中的。

当程序中用到哪一个变量的值时，由控制器发出指令将内存中该变量的值送到运算器中。经过运算器进行运算，如果需要存数，再从运算器将数据送到内存存放。



如果有一些变量使用频繁（例如在一个函数中执行 10000 次循环，每次循环中都要引用 i），为提高执行效率，C 语言允许将局部变量的值放在 CPU 中的寄存器中，需要用时直接从寄存器取出参加运算，不必再到内存中去存取。

由于对寄存器的存取速度远高于对内存的存取速度，因此可以提高执行效率。

这种变量叫做寄存器变量，用关键字 `register` 作声明。

例8·19使用寄存器变量

```
long fac(long n)
{register long i,f=1; /*拟定义寄存器变量拟*/
for (i=1;i<=n;i++)
    f=f*i;
return (f);
}
```

8.9.5 用**extern**声明外部变量

1. 在一个文件内声明外部变量

例8·20 用**extern**声明外部变量，扩展它在程序文件中的作用域。

```
#include <stdio.h>
void main()
{ int max(int,int);
  extern A,B; /*外部变量声明*/
  printf("%d\n",max(A,B));
}
int A=13,B=-8;           /*定义外部变量*/
int max(int x,int y)     /*定义max函数*/
{return(x>y?x:y);}
}
```

2. 在多文件的程序中声明外部变量

文件file1.c 中的内容为：

```
#include <stdio.h>
int A;                  /* 定义外部变量 */
void main()
{
    . . .
    c = A * b ;
    . . .
}
```

文件file2.c 中的内容为：

```
extern int A ; /* 声明A为一个已定义的外部变量 */
int power (int n) ;
{
    ...
    y *= A ;
    ...
}
```

8.9.6 用static声明外部变量

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

例如：

file1.c

```
static int A;  
void main ( )  
{  
...  
}
```

static对局部变量和全局变量的作用不同：

对局部变量来说,它使变量由动态存储方式改变为静态存储方式。

对全局变量来说,它使变量局部化(局部于本文件),仍为静态存储方式。

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

§8.10 内部函数和外部函数

§8.10.1 内部函数

如果一个函数只能被本文件中其他函数所调用,它称为内部函数。在定义内部函数时,在函数名和函数类型的前面加static。

如 **static int fun (int a , int b)**

§8.10.2 外部函数

在定义函数时，如果在函数首部的最左端加关键字 **extern**，则表示此函数是外部函数，可供其他文件调用。如：**extern int fun (int a, int b)**。

C语言规定，如果在定义函数时省略**extern**，则隐含为外部函数。本书前面所用的函数都是外部函数。

1 内容小结

§ 8.1 概述

§ 8.2 函数定义的一般形式

§ 8.3 函数参数和函数的值

§ 8.4 函数的调用

§ 8.5 函数的嵌套调用

§ 8.6 函数的递归调用

§ 8.7 数组作为函数参数

§ 8.8 局部变量和全局变量（放在里面还是外面）

§ 8.9 变量的存储类别（静态，动态，寄存器，自动，
局域，extern全局，static 声明本文件局域的全局
）

§ 8.10 内部函数和外部函数

本章作业：不分单复数： 4, 10, 13, 17

重要的事说三遍：不分学号单复！不分学号单复！不分学号单复！

(自学作业)自学“宏定义”，用带参数与不带参数的 #define、条件编译各写一个程序

补充题：

写递归程序，任意给n，能遍历数 1,2,3, … n 所有可能的全排列。对 n=8，试打印所有排列；并统计排列的个数（可用全局变量或指针统计），看是否为 8! 。

写递归程序，遍历搜索所有 3 阶、4阶幻方（行列对角线之和相等）。4阶有7000多种。

提示：可在已经生成的元素恰好能构成1行、2行、.. 时，提前判断。如果当前行之和不满足条件即不再继续生成新元素；对4阶幻方，中央四个数字之和恰好也等于相同的数。这样可提前排除很多，几千种4阶幻方能在5分钟内找完。否则遍历20亿天荒地老。

写递归程序，遍历搜索所有 3 阶、4 阶幻方（行列对角线之和相等）。

4 阶有 7000 多种。

```
    printf("%3d ",list[i]);
    if((i+1)%m==0) printf("\n");
}
printf("\n");
}

/* 寻找所有幻方 */
void HuanFang(register int m)
{
    int i, b[N], no;
    for(i=0;i<=n-1;i++)
        for(j=0;j<=k-1;j++)
            for(l=0;l<n;l++)
                if(b[i]==0)
                {
                    list[k]=l;
                    /* 提前剪枝 */
                    if(k!=0)
                        no++;
                    if(no>m)
                        if(m==1)
                            Configuration: HuanFang
[s], 0 warning(s)
The 4566-th result:
11  4  13  6
2   14  3  15
5   9  8  12
16  7  10  1
The 4567-th result:
11  4  13  6
5   7  12  10
16  14  1  3
2   9  8  15
The 4568-th result:
11  4  13  6
5   14  3  12
2   9  8  15
16  7  10  1
The 4569-th result:
11  4  13  6
5   14  3  12
8   15  2  9
10  1  16  7
The 4570-th result:
11  4  13  6
7   5  10  12
14  16  3  1
```

第九章

预处理命令

基本概念

- ANSI C标准规定可以在C源程序中加入“预处理命令”，以增强功能。
- 预处理命令不是C语言本身的组成部分，编译程序也不能识别它们；因此要在编译前先对它们“预处理”。
- C语言与其他高级语言的一个重要区别是可以使用预处理命令。

基本概念

C提供三种预处理功能：

- 1 · 宏定义
- 2 · 文件包含
- 3 · 条件编译

为了与一般C语句相区别，这些命令以符号“#”开头。例如：

#define

#include

§9.1 宏定义

9.1.1 不带参数的宏定义

宏定义一般

#**define** 标识符 字符串

形式为：

例如：# **define** PI 3.1415926

- #**define**是宏定义命令。在程序文件中用标识符PI来代“3.1415926”这个字符串，在编译预处理时，将程序中在该命令以后出现的所有PI都用它代替，使用户能以一个简单的名字代替一个长的字符串。
- 这个标识符（名字）称为“宏名”；预编译时将宏名替换成字符串的过程称为“宏展开”。
- 一般习惯（非语法规则）：宏名用全大写字母，全局变量用首字母大写。

例9.1 使用不带参数的宏定义

```
#include <stdio.h>
#define PI 3.1415926
void main()
{float l,s,r,v;
printf("input radius:");
scanf("%f",&r);
l=2.0*PI*r;
s=PI*r*r;
v=4.0/3*PI*r*r*r;
printf("l=%10.4f\ns=%10.4f\nv=%10.4f\n",l,s,v);
}
```

运行情况如下：

input radius: 4

1=25.1328

s=50.2655

v=150.7966

说明：

- (1) 宏名一般习惯用大写字母表示，以便与变量名相区别。但这并非规定，也可用小写字母。
- (2) 使用宏名代替一个字符串，可以减少程序中重复书写某些字符串的工作量。
- (3) 宏定义是用宏名代替一个字符串，只作简单置换，不作正确性检查。只有在编译已被宏展开后的源程序时才会发现语法错误并报错。

说明：

- (4) 宏定义不是C语句，不必在行末加分号。如果加了分号则会连分号一起进行置换。
- (5) #**define**命令出现在程序中函数的外面，有效范围为定义命令之后到本源文件结束。
通常，#**define**命令写在文件开头，在此文件范围内有效。
- (6) 对程序中用双撇号括起来的字符串内的字符，即使与宏名相同，也不进行置换。
- (7) 可以用#**undef**命令终止宏定义的作用域。

```
#define G 9.8  
void main()  
{  
...  
}  
  
#undef G  
  
f1()  
{  
...  
}
```

G的有效范围



在f1函数中，G不再代表9.8。这样可以灵活控制宏定义的作用范围。

说明：

(7) 在进行宏定义时，可以引用已定义的宏名，可以层层置换。

例9.2 在宏定义中引用已定义的宏名

```
#include <stdio.h>
#define R 3.0
#define PI 3.1415926
#define L 2*PI*R
#define S PI*R*R
void main()
{
    printf("L=%f\nS=%f\n",L,S);
}
```

运行情况如下：

L=18.849556

S=28.274333

9.1.2 带参数的宏定义

一般形式为：

#define 宏名 (参数表) 字符串

字符串中包含在括弧中所指定的参数

例：

```
#define S(a,b) a*b  
j  
area=S(3,2);
```

- 程序中用3和2分别代替宏定义中的形式参数a和b，用 $3 * 2$ 代替 $S(3,2)$ 。因此赋值语句展开为：

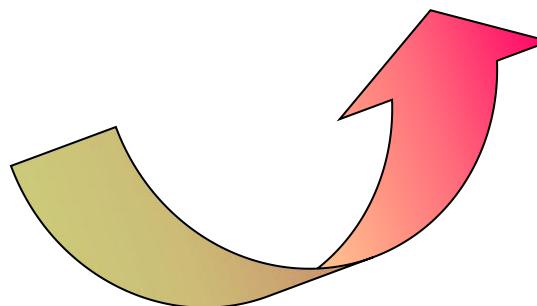
area=3*2

对带参的宏定义是这样展开置换的：

- 按照define括号中的参数
，从左到右置换。
- 串中匹配形参的字符（如
a、b）被替换。
- 串中不是参数的字符（如
a*b 中的*号）被保留。

```
# define S(a,b) a * b
```

area = S(3,2); 得 3 * 2



例9.3 使用带参的宏

运行情况如下：

```
#include <stdio.h>
#define PI 3.1415926
#define S(r) PI*r*r
void main()
{float a,area;
 a=3.6;
 area=S(a);
 printf("r=%f\narea=%f\n",a,area);
}
```

r =3.600000

area=40.715038

赋值语句 “area=S(a);” 经宏展开后为：

area=3.1415926* a * a ;

说明：

宏名与带参数的括弧之间不应加空格，否则空格以后的字符都将作为替代字符串的一部分。

带参数的宏和函数的区别：

- (1) 函数调用在程序运行时处理，为形参分配临时内存单元；宏展开则是在编译前进行的，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。
- (2) 对函数中的实参和形参类型要求一致。而宏名无类型，它的参数也无类型，只是一个符号代表，展开时代入指定的字符串即可。
- (3) 调用函数只可得到一个返回值，而用宏可以设法得到几个结果。

例9.4 通过宏展开得到若干个结果

```
#include <stdio.h>
#define PI 3.1415926
#define CIRCLE(R,L,S,V)
    L=2*PI*R;S=PI*R*R;V=4.0/3.0*PI*R*R*R
void main()
{
    float r,l,s,v;
    scanf("%f",&r);
    CIRCLE(r,l,s,v);
    printf("r=%6.2f,l=%6.2f,s=%6.2f,v=%6.2f\n",
    r,l,s,v);
}
```

带参数的宏和函数的区别：

- (4) 使用宏次数多时，宏展开后源程序长，因为每展开一次都使程序增长，而函数调用不会使源程序变长。
- (6) 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（分配单元、保留现场、值传递、返回）。

如果善于利用宏定义，可以实现程序的简化。

例9.5 通过宏展开得到若干个结果

```
#include <stdio.h>
#define PR printf
#define NL "\n"
#define D "%d"
#define D1 D NL
#define D2 D D NL
#define D3 D D D NL
#define D4 D D D D NL
#define S "%s"
```

```
void main()
{ int a,b,c,d;
char string[]="CHINA";
a=1;b=2;c=3;d=4;
PR(D1,a);
PR(D2,a,b);
PR(D3,a,b,c);
PR(D4,a,b,c,d);
PR(S,string);
}
```

运行时输出结果：

```
1
1 2
1 2 3
1 2 3 4
CHINA
```

举例：宏定义得到数组长度

```
#include <stdio.h>
#include <stdlib.h>
#define getArraylength(array) ((sizeof(array))/(sizeof(array[0])))

int main()
{
    int a[100];
    printf("%d\n",getArraylength(a));
    system("pause");
}
```

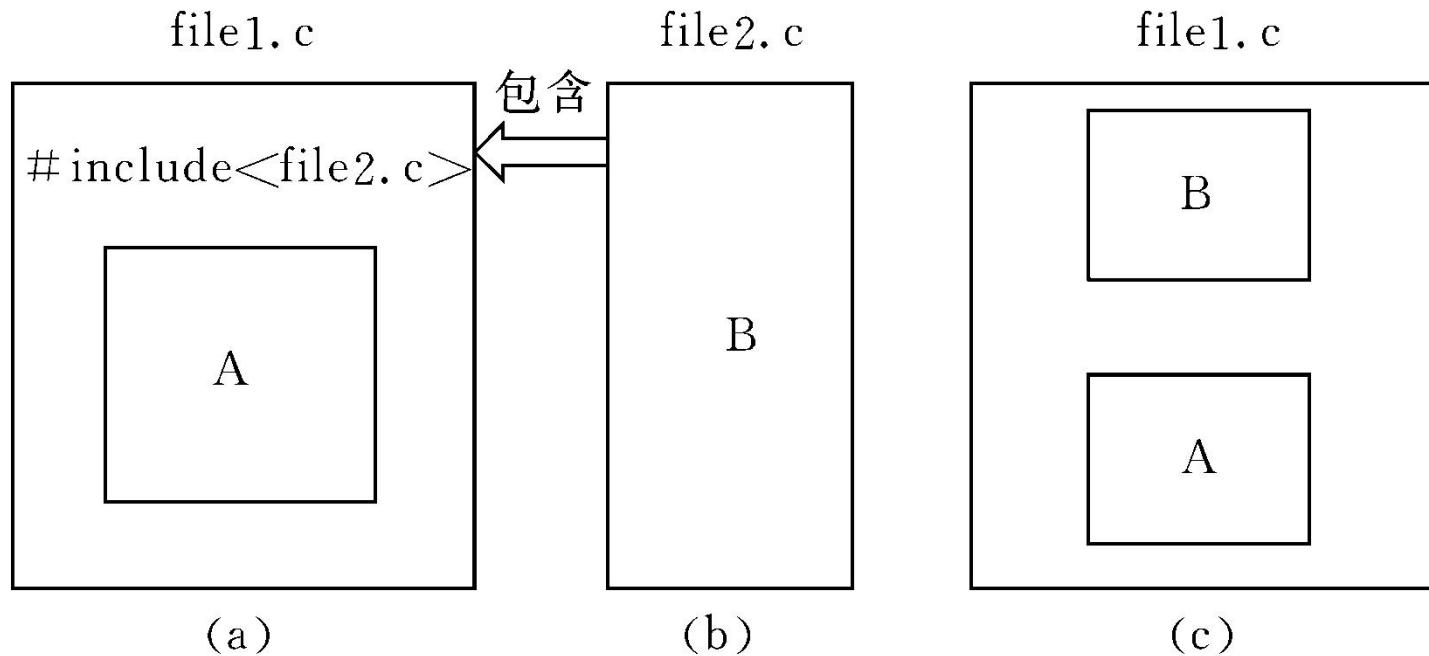
§9.2 “文件包含” 处理

- 所谓“文件包含”处理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。

其一般形式为：

#include "文件名"

或 #include <文件名>



例9.6 将例9·5时格式宏做成头文件，把它包含在用户程序中。

(1)将格式宏做成头文件

```
format.h  
#include <stdio.h>  
#define PR printf  
#define NL "\n"  
#define D "%d"  
#define D1 D NL  
#define D2 D D NL  
#define D3 D D D NL  
#define D4 D D D D NL  
#define S "%s"
```

(2)主文件file1.c

```
#include <stdio.h>  
#include "format.h"  
void main()  
{ int a,b,c,d;  
char string[]="CHINA";  
a=1;b=2;c=3;d=4;  
PR(D1,a);  
PR(D2,a,b);  
PR(D3,a,b,c);  
PR(D4,a,b,c,d);  
PR(S,string);  
}
```

注意：

编译时并不是分别对两个文件分别进行编译，而是在经过预处理后将头文件format.h包含到主文件中，得到一个新的源程序，然后对这个文件进行编译。

说明：

- (1) 一个#include命令只能指定一个被包含文件，如果要包含n个文件，要用n个#include。

- (2) #include命令中，文件名可以用双撇号或尖括号括起来。

- (3) 在一个被包含文件中又可以包含另一个被包含文件，即文件包含是可以嵌套的。

§9.3 条件编译

概念：所谓“条件编译”，是对部分内容指定编译的条件，使其只在满足一定条件才进行编译。

条件编译命令的几种形式：

(1) #ifdef 标识符
 程序段 1
#else
 程序段 2
#endif

(2) #ifndef 标识符
 程序段 1
#else
 程序段 2
#endif

(3) #if 表达式
 程序段 1
#else
 程序段 2
#endif

例9·7 输入一行字母字符，根据需要设置条件编译，使之能将字母全改为大写输出，或全改为小写字母输出。

```
#include <stdio.h>
#define LETTER 1
void main()
{char str[20]={"C Language",c;
    int i;
    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);
    }
}
```

运行结果为：

C LANGUAGE

问题：上述问题用if语句也可以实现，那用条件编译有什么好处？

- (1) 目标程序短，EXE体积小（不是所有语句都编译）
- (2) 运行时间短（程序运行时不会有太多检测if语句）

两点说明

(1) `return` 语句的功能

终止函数运行，返回上一级函数，如：

```
return;
```

也可以带回一个返回值，如：

```
return 1;
```

```
return(x+y);
```

(2) 请把每次作业做成一个文件，分别整理好 ，可以在主函数中一句话调用。（示例）

养成良好的习惯。

第十章

指針

§10.1地址和指针的概念

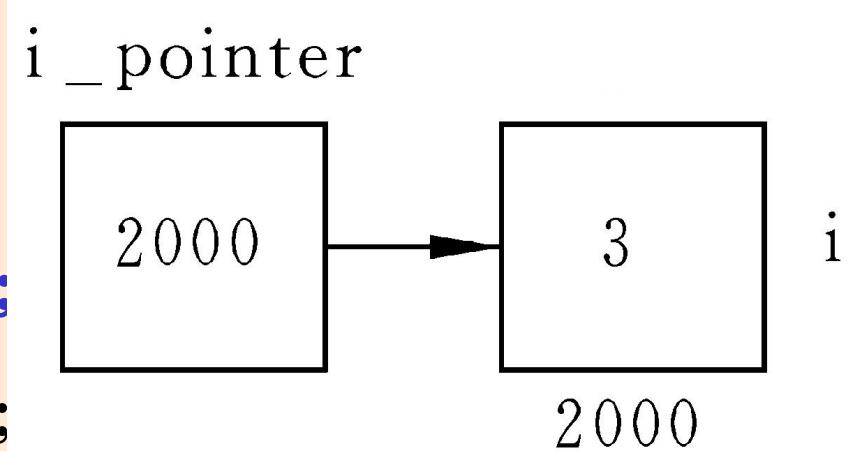
在C语言中，指针是一种特殊的变量，它是存放地址的。

例：下面定义了一个指针变量*i_pointer*，专门存放整型变量的地址。我们让它存放整型变量*i*的地址（俗话说，“*i_pointer* 指向 *i*”）：

```
int i;  
int *i_pointer;  
i_pointer = & i ;
```

用 * 声明变量的类型为指针；

&为取地址运算符。



下面都是合法的定义：

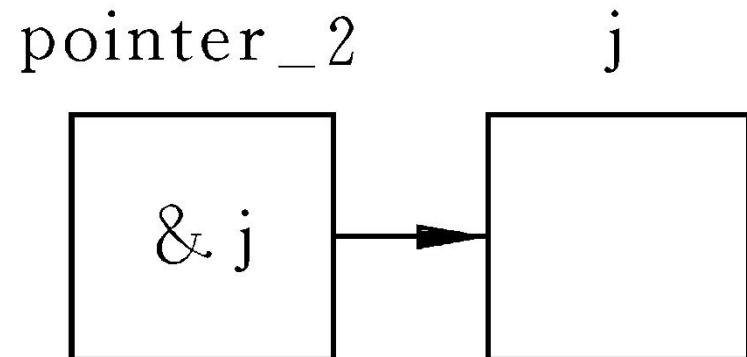
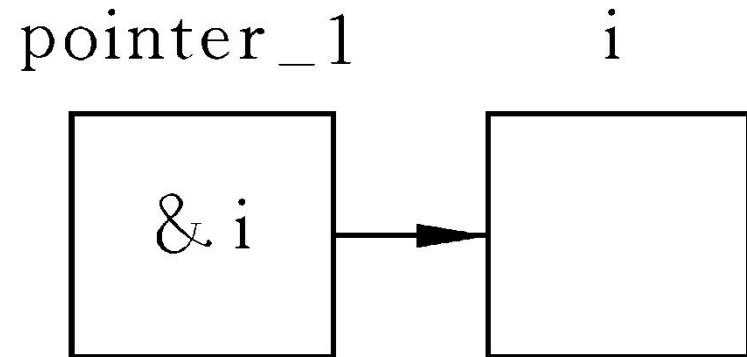
```
float *pointer_3 ; // 指向float型变量的指针变量
```

```
char *pointer_4 ; //指向字符型变量的指针变量
```

赋值方式：

```
pointer_1 = & i ;
```

```
pointer_2 = & j ;
```



注意：

(1) “*” 表示要定义变量的为指针型变量。

对 `float *pointer_1;` 指针变量名是 `pointer_1`，而不是 `* pointer_1`。

(2) 必须指定基类型。

(3) 赋值时类型必须匹配。

下面的赋值是错误的：

```
float a;
```

```
int * pointer_1;
```

```
pointer_1=&a;
```

/* 将float型变量的地

址放到指向整型变量的指针变量中，错误 */

10.2.2 指针变量的引用

例10.1 通过指针变量访问整型变量

```
int a = 100, b = 10 ;
int *pointer_1, *pointer_2 ;

pointer_1 = &a ; /*把变量 a 的地址赋给
                  pointer_1 */
pointer_2 = &b ; /*把变量 b 的地址赋给
                  pointer_2 */

printf ("%d , %d\n", a, b) ;
printf ("%d , %d\n", *pointer_1, *pointer_
2 )
```

“*”运算符的两种意义

```
int a;  
int * pointer_1;  
pointer_1 = & a;  
* pointer_1 = 3;
```

int * 中， * 说明p是一个指针变量。

* pointer_1 =3 中， * 号表示取p所指向的内存单元。所以这句话等价于 a=3。

对“`&`”和“`*`”运算符说明：

如果已执行了语句 `pointer_1 = & a ;`

(1) `& * pointer_1` 的含义？

答案：与 `& a` 相同，即变量 `a` 的地址。

(2) `* & a` 的含义？

答案：就是变量 `a`。

(3) `(*pointer_1) ++` ?

答案：相当于 `a ++`。

注意括号是必要的，如果没有括号，就成为了 `*(pointer_1++)`。使 `pointer_1` 的值改变，不再指向 `a` 了。

10.2.3 指针变量作为函数参数

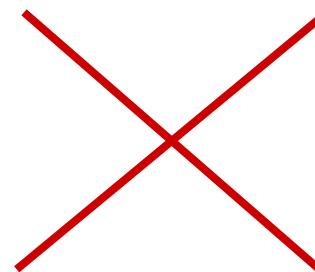
例：交换两个整数的大小

```
void swap (int *p 1 , int *p 2 )  
{  int temp ;  
    temp = *p 1 ;  
    *p 1 = *p 2 ;  
    *p 2 = temp ;  
}
```

调用举例：int a=1,b=2; swap(&a,&b);

作为对比，如果不使用指针，就无法改变实参的值

```
void swap (int p 1 ,int p 2 )  
{ int temp ;  
temp=p 1 ;  
p 1 =p 2 ;  
p 2 =temp ;  
}
```



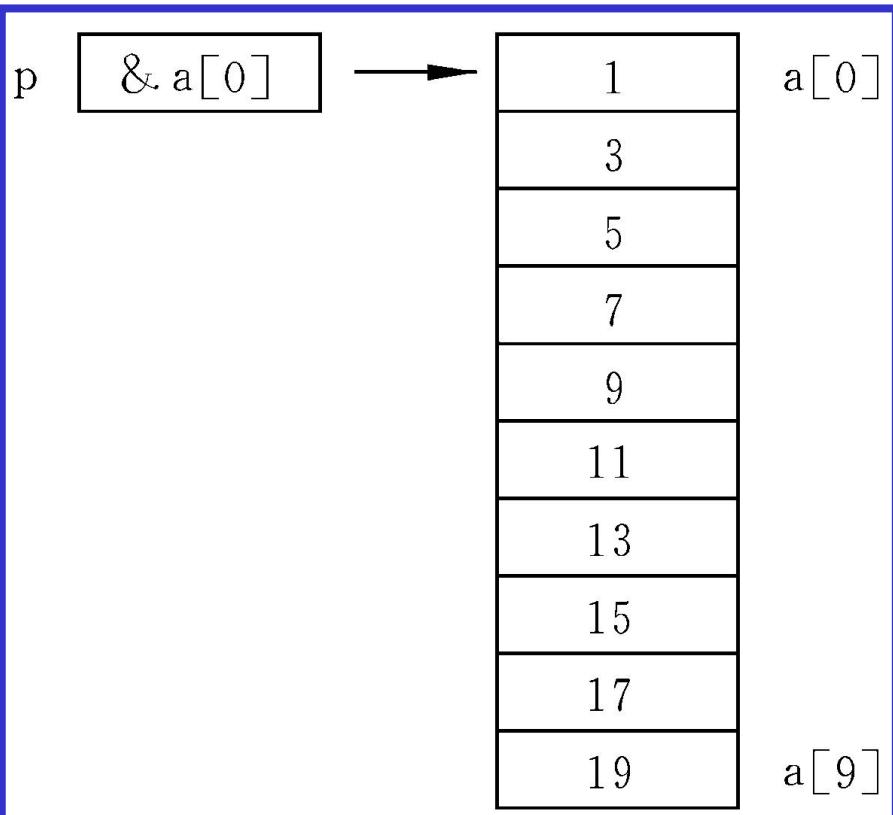
该函数无法实现 **p1**、**p2** 数值的交换！！！

§10.3 数组与指针

指针变量当然也可以指向数组元素，如

int a[10]; int *p; p=&a[0];

使 p 指向 a 数组的第 0 号元素，如图：



注意：**C**语言中数组名的值为数组的首地址。类似于“常量指针”。

以上两个语句等价：
p=&a[0];
p=a;

10.3.2 通过指针引用数组元素

- (1) 下标法，如 $a[i]$ ， $p[i]$ 形式；
- (2) 指针法，如 $*(\mathbf{a} + i)$ 或 $(*p + i)$ 。

这里 $\mathbf{a} + i$ ， $p + i$ 表示正好对应 $a[i]$ 的地址。

提问： $(\mathbf{a} + i)$ 的含义？

回答：就是从首地址移动 i 个 int 类型的字节数。因此得到第 i 个元素的地址，即 $\&a[i]$

折磨你： $(\mathbf{a} + i)[j]$ 的含义？

回答：从 $\mathbf{a} + i$ 再向后移动 j ，然后取值。所以是 $a[i + j]$ 或者 $(*(\mathbf{a} + i + j))$

小结数组与指针的替换规则：

$\textcircled{S}[i]$ 等价于 $*(\textcircled{S} + i)$ 。

whatever \textcircled{S} , no matter how
complicated \textcircled{S} ...

10.3.3 用数组名作函数参数

之前介绍过，如：

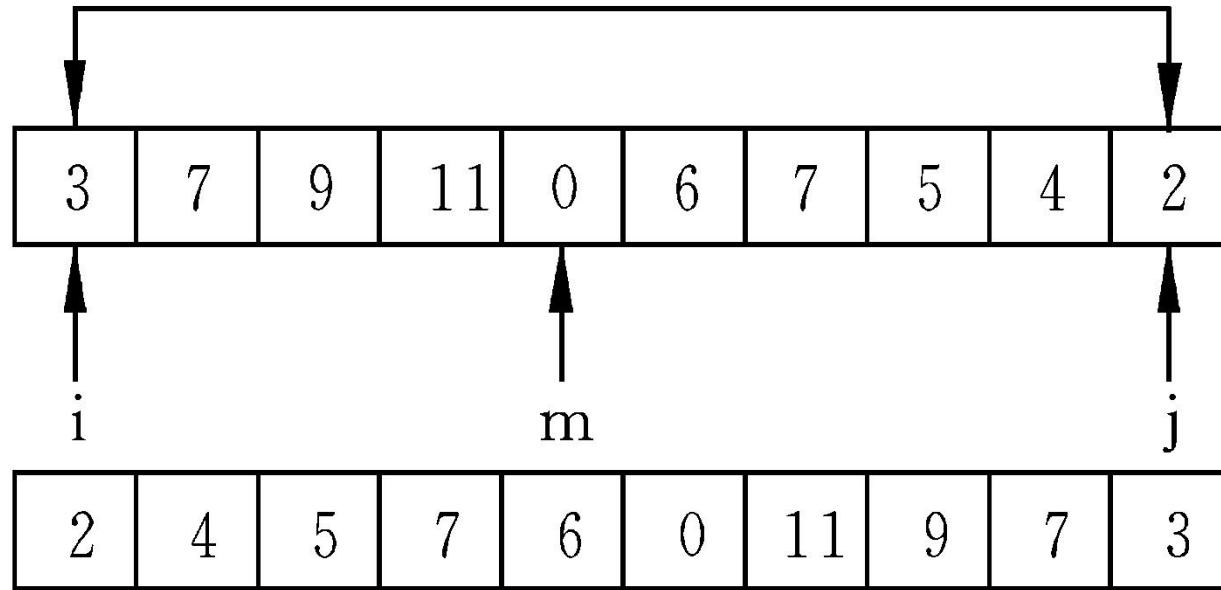
```
void f(int arr [ ] , int n )  
{ ... }
```

编译时是将arr按指针变量处理的，因此声明时也可以写成

```
void f (int *arr, int n)
```

以上两种写法等价。

例10·7 将数组a中n个整数按相反顺序存放



如何用指针实现？？？

```
void inv_array(int *arr, int n){  
    int *p1, *p2, *pmid;  
    int tmp;  
    p1=arr; p2=arr+n-1; pmid=arr+n/2;  
    while(p1<pmid){  
        tmp=*p1;  
        *p1=*p2;  
        *p2=tmp;  
        p1++; p2--;  
    }  
}
```

下面举例，说明使用指针实现 **copy_string** 函数

1、**copy_string**函数可写为

```
void copy_string (char * from, char * to)
{ while ( (*to = *from) != '\0')
    { to++; from++; }
}
```

函数体还可改为

```
while ( (*to++ = *from++) != '\0') ;
```

字符数组和字符指针的区别

- (1) 字符数组由若干个元素组成，而字符指针存放的是字符串第1个字符的地址，决不是将字符串放到字符指针变量中。
- (2) 字符数组只能对各个元素赋值，不能用以下办法：

```
char str [ 1 4 ] ;
```

```
str = " I l o v e C h i n a ! " ;
```

而对字符指针可以采用下面方法赋值：

```
char * a ;
```

```
a = " I l o v e C h i n a ! " ;
```

但注意赋给 a 的不是字符，而是字符串第一个元素的地址。

(3) 如果定义了一个字符数组，在编译时为它分配内存单元，有确定的地址。

而定义一个字符指针变量时，并未具体指向确定的内存地址。

如：**char str [10] ;**

scanf ("%s", str) ; 是可以的。

而常有人用下面的方法输入一个字符串，虽然也能运行，但却是危险的（a指向随机未知内存！）：

char * a ;

scanf ("%s", a) ;

10.3.4 多维数组与指针

1. 多维数组元素的地址

二维数组是“数组的数组”，例：

定义

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

第一个元素 $a[0]$ 为数组 {1, 3, 5, 7}

利用 $*(a[0] + j)$ 就可以引用这四个数字的值

a 是“指向长度为四的一维数组的指针”

用“ $a + i$ ”就可以引用 $a[0]$ $a[1]$ $a[2]$ 这三个数组

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

表示形式	含义	地址
a	二维数组名，指向一维数组a[0]，即0行首地址	2000
a[0], *(a+0), *a	0行0列元素地址	2000
a+1, &a [1]	1行首地址	2008
a [1], *(a+1)	1行0列元素a[1][0]的地址	2008
a[1]+2, *(a+1)+2, &a[1][2]	1行2列元素a[1][2] 的地址	2012
*(a[1]+2), *(*(a+1)+2), a[1][2]	1行2列元素a [1] [2] 的值	元素值为13

- 提问：获得 a 第1行，第2列的元素的值共有哪些方式？？？
- 答：
 - 当然可以用 $a[1][2]$
 - 也可以用 $*(a[1]+2)$ ： $a[1]$ 的第二个元素
 - 因为 $a[1]$ 与 $*(a+1)$ 等价，因此替换到 $a[1][2]$ ，写成：
 - $(*(a+1)) [2]$
 - 这里 $*(a+1)$ 必须加括号。[] 的优先级比*高，因此 $*(a+1)[2]$ 等于 $*((a+1)[2])$ 等于 $*(a[3])$ 等于 $a[3][0]$
 - 也可以全用指针： $*(*(a+1)+2)$

小结数组与指针的替换规则：

$\textcircled{S}[i]$ 等价于 $*(\textcircled{S} + i)$ 。

whatever \textcircled{S} , no matter how
complicated \textcircled{S} ...

• 折磨你：对二维数组 $a[3][4]$ ，我们想获得它第*i*行、第*j*列元素的值。请问有哪几种方式？

$a[i][j]$

把后面的 $[j]$ 替换掉： $*(a[i]+j)$

把前面的 $[i]$ 替换掉： $(*(a+i))[j]$

两个都替换掉： $*(*(a+i)+j)$

还可把a等价成一维数组处理： $(\&a[0][0])[i*4+j]$

把上面的 $[]$ 也替换掉： $*((\&a[0][0])+i*4+j)$

请自己测试上面的各种方式，是否都得到想要的结果。

- 继续折磨：函数中使用二维数组作变量，可以这样写：

- `void fun(int a[][3], int n)`

- 如果用指针写，该怎么写？。。。

- 回答：

- `a[]` 就用 `*a` 替换，因此是

- `void fun(int (* a)[3], int n)`

- 此时 `a` 是指针，指向“长度3的1维数组”

- 升级虐待：要函数返回一个二维数组？😊
如： `fun(int a[3][3])`，返回 `a` 的转置。

- 答：

- 此时 `fun` 是指针，指向长度3的数组，也就是 `fun[][3]`。因此替换为：`int (*fun)[3]`。
- 因此声明函数时，就整个替换写好，成为：
`int (*fun(a[3][3])) [3]`
- 注意：此例纯属游戏，绝不提倡这种反人类的写法。使用
`void fun(int a[][3], b[][3])`
其中 `b` 保存转置的结果，也能实现同样的功能。
- 返回二维数组实际应用中用处不大。实在要返回，用结构体也更容易。

10.4 void指针类型

ANSIC新标准增加了一种“void”指针类型，即可定义一个指针变量，但不指定它是指向哪一种类型数据的。在将它的值赋给另一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 ;  
void *p2 ;  
...  
p1 = (char *) p2 ;
```

同样可以用 (void *) p1 将 p1 的值转换成 void * 类型。如：

```
p2 = (void *) p1 ;
```

malloc 实现“动态数组”

(动态数组，是不是盼得两眼汪汪？)

void *malloc(long NumBytes)

- 该函数分配了NumBytes个字节，并返回了指向这块内存的指针。
如果分配失败，则返回一个空指针（NULL）。
- 关于分配失败的原因，应该有多种，比如说空间不足就是一种。

void free(void *FirstByte)

- 该函数是将之前用malloc分配的空间还给程序或者是操作系统，也就是释放了这块内存，让它重新得到自由。

malloc 用法

其实这两个函数用起来倒不是很难，也就是malloc()之后觉得用够了就用了它把它给free()了，举个简单例子：

```
char *p = NULL;
```

```
p = (char *)malloc(100 * sizeof(char)); // 强制类型转换为 (char*)
```

类型，赋值给p！此时p指向这段长度为100个char的内存空间的第一个byte！

```
if (NULL == p) exit (1);
```

```
gets(p);
```

```
...
```

```
free(p); // 释放整段内存。 注意：如果p的值通过p++等改变过（即，p不再指向内存段首地址），释放内存容易出现错误！
```

```
p = NULL; // 将 p 指向空（不指向任何内存），防止指向的内存释放之后还被使用
```

malloc 多说几句

1、malloc()到底从哪里得到了内存空间？

答案是从堆里面获得空间。也就是说函数返回的指针是指向堆里面的一块内存。

操作系统中有一个记录空闲内存地址的链表（以后再解释什么是链表）。当操作系统收到程序的申请时，就会遍历该链表，然后就寻找第一个空间大于所申请空间的堆结点，然后就将该结点从空闲结点链表中删除，并将该结点的空间分配给程序。

malloc 多说几句

堆和栈

栈（英文名称是stack）是系统自动分配空间的，例如我们定义一个

```
char a ;
```

系统会自动在栈上为其开辟空间。

堆（英文名称是heap）则是程序员根据需要自己申请的空间，例如

```
malloc (10) ;
```

开辟十个字节的空间。

栈上的空间是自动分配自动回收的，所以栈上的数据的生存周期只是在函数的运行过程中，运行后就释放掉，不可以再访问。

而堆上的数据只要程序员不释放空间，就一直可以访问到，不过缺点是一旦忘记释放会造成内存泄露。

更多内容推荐看：

<http://www.cnblogs.com/TonyEwsn/archive/2010/01/29/1659496.html>

作业

- 自觉弄懂指针的概念、使用方法，弄明白所有ppt上所有内容。
- 指针一章课后题：**1-6, 9-10**，不分单复数。说三遍：不分 不分 不分单复
- (补充题) 计算机模拟三门问题(1亿次)，显示不换门、换门获胜的概率分别为 $1/3$ 、 $2/3$ 。用指针带回两个结果。函数形式如下
 - **void sanmen(int n, int *n1, int *n2)** // 随机模拟n次，换门得胜n1次，不换门得胜n2次。
- (补充题[本题可延到下次交]) 操作任意长度二维双精度矩阵（长度在函数中作为自变量输入、是动态变量），实现以下功能：用 0-1 随机数初始化；打印；矩阵乘法；初等变换求逆(暂不考虑某元素为0的可能性)：
 - 1. 生成 $2*2$ 随机方阵，打印、求逆、打印矩阵与逆的乘积
 - 2. 同上操作 $n=3, 4, 5$ 方阵。
 - 提示：生成任意范围内随机双精度数的函数为

```
double fRand(double fMin, double fMax){  
    f=(double)rand() / RAND_MAX; // Here RAND_MAX is the  
    maximal possible value of rand()  
    return fMin + f * (fMax - fMin);
```

10.5 指向函数的指针

一个函数在编译时被分配给一个入口地址。这个**函
数的入口地址就称为函数的指针**。例如，

```
printf("%p \n", sin);
```

// %p 为打印指针的格式，按照16进制输出内存地址

执行结果为类似下面的一串地址：

004034A4

因此，可以用指针变量指向一个函数！

例10.22 求a和b中的大者。

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

void main ( )
{
    int (* p ) (int, int) ;
    // 声明指向函数的指针： 类型(* 变量名) (形参类型)
    p = m a x ;
    // 将函数的入口地址赋给指针变量。
    // 这一步只需给出函数名，不需给出参数 (understandable)
    printf("%d \n", (* p ) (1 , 2) );
}

// 运行结果为 2
```

例：求 **sin, cos, exp, sinh, cosh** 中任意两个作用到1.0时的结果。

```
double (*pfuns[5])(double) = {sin, cos, exp, sinh, cosh};  
                                // 声明了五个指向函数的指针构成的数组  
char funnames[5][10] = {"sin", "cos", "exp", "sinh", "cosh"};  
double tmpx;  
  
for(int i=0;i<5;i++)  
    for(int j=0;j<5;j++){  
        tmpx = (*pfuns[j])(1.0); //作用第一个函数  
        tmpx = (*pfuns[i])(tmpx); //作用第二个函数  
        printf("\t%s \t%s \t( 1.0 ) \t= \t%f \n", funnames[i], funnames[j],  
tmpx); //打印结果  
    }
```

结果见右。

一个复杂的函数
组合问题，借助指向
函数的指针很容易就
实现了。

"C:\Users\pcfly\Desktop\教学 高级算法与编程\Code\Debug\Chap8_pointer.exe"

sin	sin	(1.0)	=	0.745624
sin	cos	(1.0)	=	0.514395
sin	exp	(1.0)	=	0.410781
sin	sinh	(1.0)	=	0.922767
sin	cosh	(1.0)	=	0.999616
cos	sin	(1.0)	=	0.666367
cos	cos	(1.0)	=	0.857553
cos	exp	(1.0)	=	-0.911734
cos	sinh	(1.0)	=	0.385357
cos	cosh	(1.0)	=	0.027712
exp	sin	(1.0)	=	2.319777
exp	cos	(1.0)	=	1.716526
exp	exp	(1.0)	=	15.154262
exp	sinh	(1.0)	=	3.238795
exp	cosh	(1.0)	=	4.678982
sinh	sin	(1.0)	=	0.944350
sinh	cos	(1.0)	=	0.566977
sinh	exp	(1.0)	=	7.544137
sinh	sinh	(1.0)	=	1.465019
sinh	cosh	(1.0)	=	2.232630
cosh	sin	(1.0)	=	1.375426
cosh	cos	(1.0)	=	1.149549
cosh	exp	(1.0)	=	7.610125
cosh	sinh	(1.0)	=	1.773776
cosh	cosh	(1.0)	=	2.446352

函数指针变量常用的用途之一是把函数的入口地址作为参数传递到其他函数。

常用于微分、积分、解方程、插值、求最小值、拟合回归、傅里叶变换等通用功能。操作的对象是任意给定的函数。函数指针在结构化编程中是经常使用的。

```
void fun (int (*f1)(int) , int (*f2)(int,int))
{
    int a , b , i , j ;
    a = (*f1) ( i ) ; /*调用 f 1 函数*/
    b = (*f2) ( i , j ) ; /*调用 f 2 函数*/
    ...
}
```

10.6 返回指针值的函数

一个函数可以带回一个整型值、字符值、实型值等，也可以带回指针型的数据，即地址。

概念与以前类似，只是带回的值的类型是指针类型而已。

一般定义形式为

类型名 *函数名（参数表列）；

例如：

int *a (int x, int y);

可用于，如：返回数组最大元素的地址；...

例10·24 有若干个学生的成绩（每个学生有4门课程），要求在用户输入学生序号以后，能输出该学生的全部成绩。用指针函数来实现。

```
#include <stdio.h>
void main ()
{ float *score [ ] [4] ={{60, 70, 80, 90},
                         {56, 89, 67, 88}, {34, 78, 90, 66}};
float search (float (*pointer) [4], int n) ;
float *p ;
int i , m ;
printf ("enter the number of student :") ;
scanf ("%d", &m) ;
printf ("The scores of No · %d are :\n", m) ;
```

```
p=search ( s c o r e ,m) ;
for ( i =0 ; i <4 ; i ++=
printf ("%5 .2 f \t",*( p +i ) ) ;
}

float * search ( float (*p o i n t e r )[4] ,int
n )
{ float *p t ;
p t =*( p o i n t e r +n ) ;
return ( p t ) ;
}
```

运行情况如下：

enter the number of student : 1 ↴ 疑

The scores of No. 1 are :

56.00 89.00 67.00 88.00

例10·25 对上例中的学生，找出其中有不及格课程的学生及其学生号。

```
#include <stdio.h>
void main ()
{float score [ ] [4] ={{60, 70, 80,
90}, {56,
89, 67, 88}, {34, 78,
90, 66}};
    float search (float (*pointer) [4] ) ;
    float *p ;
    int i , j ;
```

```
for ( i = 0 ; i < 3 ; i ++ )
{ p = search ( score + i ) ;
  if ( p == * ( score + i ) )
    { printf ("No. %d scores : " , i ) ;
      for ( j = 0 ; j < 4 ; j ++ )
        printf ("%5.2f " , * ( p + j ) ) ;
      printf ("\n") ;
    }
  }
```

10.7 指针数组和指向指针的指针

10.7.1 指针数组的概念

一个数组，若其元素均为指针类型数据，称为指针数组，数组中的每一个元素都是一个指针变量。一维指针数组的定义形式为：

类型名 约数组名 [数组长度] ;

int *p[4];

// 举例：指向整型变量的指针的数组

// 不能写成 int (*p)[4] // 指向数组的指针

前面我们已经展示了指向函数的指针数组 double (*pfun[5])(double)。

10.7 指针数组和指向指针的指针

指针数组比较适合用来指向若干个字符串。

例如，图书馆有若干本书，想把书名放在数组里，然后对这些书名进行排序和查询。

按一般方法，字符串本身就是一个字符数组，因此要设计一个二维字符数组才能存放多个字符串；但在定义二维数组时，需要指定列数。而实际上各个书名是不一样长的。如果按照最长的书名定义，就会浪费很多内存单元。

10.7 指针数组和指向指针的指针

因此可以定义一些字符串，然后用指针数组的元素分别指向各个字符串。如：

```
char * name[] = {"C", "Quantum Mechanics", "Machine  
learning", "Parallel Programming in C with MPI and OpenMP"};  
// 最后一本书名字很长。指针数组的定义为存储《C》的书名节省了空  
间。  
for(int i=0; i<4; i++) puts(name[i]);
```

10 · 7 · 2 指向指针的指针

怎样定义一个指向指针数据的指针变量呢？如下：

```
char **p ;
```

p 的前面有两个*号。*运算符的结合性是从右到左，因此**p 相当于* (*p)。

前面有两个*号，表示指针变量 p 是指向一个字符指针变量的。

*p 是指向字符的指针，p 是指向字符指针的指针。

例10·27 使用指向指针的指针

```
#include <stdio.h>

void main ( )  {
    char *name [ ] = {"Follow me", "BASIC", "Great
Wall", "FORTRAN", "Computer design"};
    // 声明指针数组；

    char **p ; // 声明指向指针的指针；

    for (p=name ; p<name+5 ; p++) puts(*p);
    // 移动指向指针的指针，逐个打印字符串
}
```

10.8 有关指针的数据类型和指针运算的小结

定义	含义
int 约*p ;	指针
int 约*p [n] ;	指针数组
int (*p) [n] ;	指向数组的指针
int 约*p () ;	返回指针的函数
int (*p) () ;	指向函数的指针
int **p ;	指向指针的指针

指针运算小结

(1) 指针变量加(减)一个整数

例如： $p++$ 、 $p--$ 、 $p+i$ 、 $p-i$ 、 $p+=i$ 、 $p-=i$ 等。

(2) 指针变量赋值

$p = \&a$ ； (将变量 a 的地址赋给 p)

$p = array$ ； (将数组 array 首元素地址赋给 p)

$p = \&array[i]$ ； (将数组 array 第 i 个元素
的地址赋给 p)

$p = max$ ； (max 为已定义的函数，将 max 的入
口

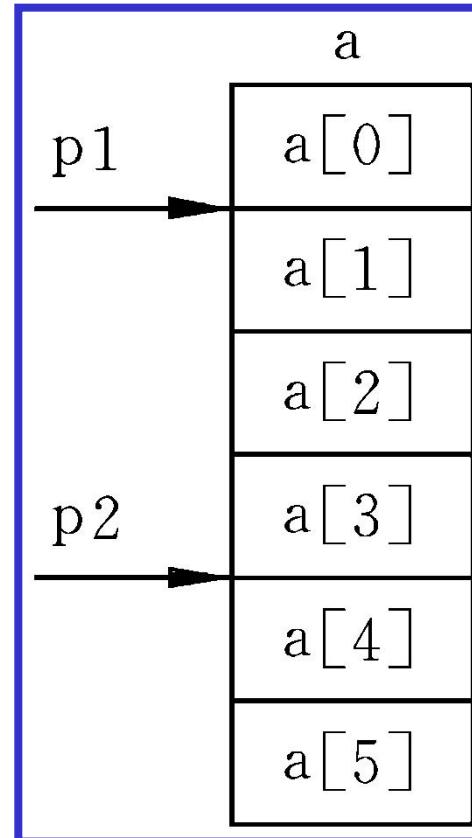
地址赋给 p)

$p1 = p2$ ； ($p1$ 和 $p2$ 都是指针变量，将 $p2$ 的
值赋给 $p1$)

(3) 指针变量可以有空值，即该指针变量不指向任何变量，可以这样表示： $p=NULL$ ；

(4) 两个指针变量可以相减

如果两个指针变量都指向同一个数组中的元素，则两个指针变量值之差是两个指针之间的元素个数



(5) 两个指针变量比较

若两个指针指向同一个数组的元素，则可以进行比较。指向前面的元素的指针变量“小于”指向后面元素的指针变量。

#####

本周作业：

13，18，20 不分单复数。

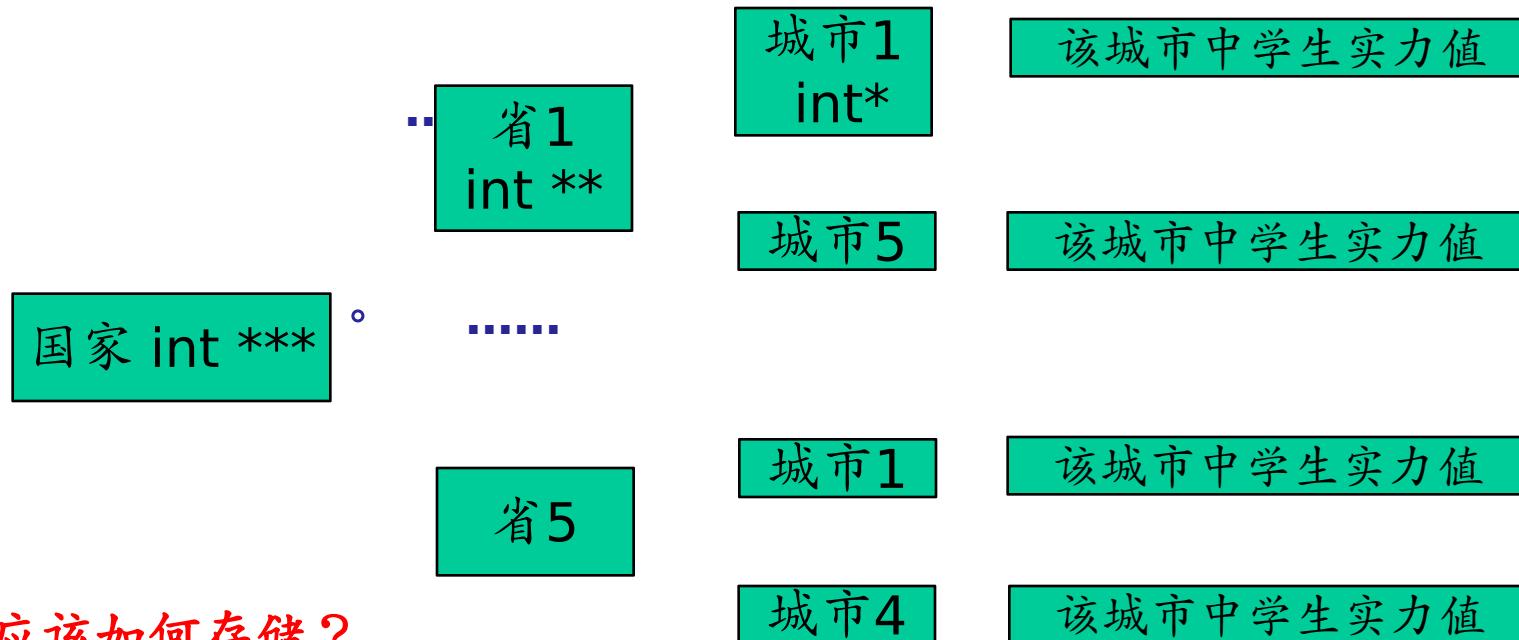
对这三道题的要求：

声明中可以出现数组，执行语句中不许出现 []

自学作业：自学位运算，完成该章的课后练习。

补充作业 (这道题用结构体更容易实现，anyway放在这里虐大家)

- 已知某个国家，有**5**个省；每个省有随机**3-5**个城市；每个城市有**5-10**名学生；每个学生实力随机（实力值**0-10000**）。请编程模拟一个全国竞赛的过程：
 - **Step 1.** 用 **malloc** 开辟动态数据结构，存储国家、省、城市、学生的信息



问：应该如何存储？

- 用 **malloc** 开辟随机长度的 **int** 型数组，存储学生；第一个元素存储学生个数，后面各个元素存储学生实力值。
- 城市是 **int *** 指针，指向该数组首元素；省是 **int **** 指针，指向第一个城市；国家是 **int ***** 指针（或 **int **** 型数组）。
- 先开辟国家，再开辟省，再开辟城市。
- 可以开辟 **int** 型、长度为**5**数组，存放每个省的城市个数。

补充作业

- 已知某个国家，有**5**个省；每个省有随机**3-5**个城市；每个城市有**5-10**名学生；每个学生实力随机（实力值**0-10000**）。请编程模拟一个全国竞赛的过程：
 - **Step 1.** 用 **malloc** 开辟动态数据结构，存储国家、省、城市、学生的信息
- 输出结果示例：

```
C:\Users\pcfly\Desktop\教学 高级算法与编程\Code\Debug\Chap8_pointer.exe
-----
***** SIMULATION BEGINS!!!! *****
-----
Province 0 ( 5 cities):
    City 0 ( 7 student)s: 9708 4002 5474 5447 4903 2056 6478
    City 1 ( 6 student)s: 9296 9784 4622 2208 2682 627
    City 2 ( 10 student)s: 1000 5341 9409 513 7154 4106 8387 7939 3365 843
    City 3 ( 7 student)s: 2601 2740 1079 4389 6771 2724 1843
    City 4 ( 9 student)s: 1151 3717 7807 2528 5023 4458 4096 3983 3625
Province 1 ( 3 cities):
    City 0 ( 10 student)s: 6348 1296 283 2374 680 7426 9201 42 9440 8241
    City 1 ( 8 student)s: 4531 7513 2338 6758 3213 5948 6855 2050
    City 2 ( 6 student)s: 8499 4883 8652 4909 1887 1991
Province 2 ( 3 cities):
    City 0 ( 10 student)s: 3302 2003 2988 4126 1733 5116 474 1249 8480 371
    City 1 ( 8 student)s: 6803 5058 2432 8114 9663 2141 1821 4031
    City 2 ( 6 student)s: 1766 4579 9565 3057 2316 9218
Province 3 ( 3 cities):
    City 0 ( 5 student)s: 8558 3763 7705 6308 1051
    City 1 ( 5 student)s: 7392 4633 3582 4151 3797
    City 2 ( 10 student)s: 839 5327 1808 2339 317 7691 1393 9567 5040 9392
Province 4 ( 3 cities):
    City 0 ( 7 student)s: 6366 2740 2403 5911 5601 5979 500
    City 1 ( 7 student)s: 5850 1517 8227 5785 2737 7258 6716
    City 2 ( 6 student)s: 7296 3993 9572 988 8691 6233
-----
PRESS TO BEGIN SIMULATION!
```

补充作业

- 已知某个国家，有**5个省**；每个省有随机**3-5个城市**；每个城市有**5-10名学生**；每个学生实力随机（实力值**0-10000**）。请编程模拟一个全国竞赛的过程：
 - **Step 2.** 先在每个城市举行比赛，选出第一名；将第一名移动到该城市学生第一位置，如下：

```
C:\Users\pcfly\Desktop\教学 高级算法与编程\Code\Debug\Chap8_pointer.exe"
PRESS TO BEGIN SIMULATION!
!-----!
EACH CITY COMPETITION:
!-----!
Province 0 ( 5 cities):
    City 0 ( 7 students): 9708 4002 5474 5447 4903 2056 6478
    City 1 ( 6 students): 9784 9296 4622 2208 2682 627
    City 2 ( 10 students): 9409 5341 1000 513 7154 4106 8387 7939 3365 843
    City 3 ( 7 students): 6771 2740 1079 4389 2601 2724 1843
    City 4 ( 9 students): 7807 3717 1151 2528 5023 4458 4096 3983 3625
Province 1 ( 3 cities):
    City 0 ( 10 students): 9440 1296 283 2374 680 7426 9201 42 6348 8241
    City 1 ( 8 students): 7513 4531 2338 6758 3213 5948 6855 2050
    City 2 ( 6 students): 8652 4883 8499 4909 1887 1991
Province 2 ( 3 cities):
    City 0 ( 10 students): 8480 2003 2988 4126 1733 5116 474 1249 3302 371
    City 1 ( 8 students): 9663 5058 2432 8114 6803 2141 1821 4031
    City 2 ( 6 students): 9565 4579 1766 3057 2316 9218
Province 3 ( 3 cities):
    City 0 ( 5 students): 8558 3763 7705 6308 1051
    City 1 ( 5 students): 7392 4633 3582 4151 3797
    City 2 ( 10 students): 9567 5327 1808 2339 317 7691 1393 839 5040 9392
Province 4 ( 3 cities):
    City 0 ( 7 students): 6366 2740 2403 5911 5601 5979 500
    City 1 ( 7 students): 8227 1517 5850 5785 2737 7258 6716
    City 2 ( 6 students): 9572 3993 7296 988 8691 6233
PRESS TO MOVE TO NEXT ROUND...
```

补充作业

- 已知某个国家，有**5**个省；每个省有随机**3-5**个城市；每个城市有**5-10**名学生；每个学生实力随机（实力值**0-10000**）。请编程模拟一个全国竞赛的过程：
 - **Step 3.** 每个省进行比赛，第一名最牛的城市排到该省第一位，如下：

```
选择"C:\Users\pcfly\Desktop\教学 高级算法与编程\Code\Debug\Chap8_pointer.exe"  
!-----  
EACH PROVINCE COMPETITION:  
!-----  
Province 0 ( 5 cities):  
  (find max, imax = 9784 1 )  
    City 0 ( 6 students): 9784  9296  4622  2208  2682  627  
    City 1 ( 7 students): 9708  4002  5474  5447  4903  2056  6478  
    City 2 ( 10 students): 9409  5341  1000  513   7154  4106  8387  7939  3365  843  
    City 3 ( 7 students): 6771  2740  1079  4389  2601  2724  1843  
    City 4 ( 9 students): 7807  3717  1151  2528  5023  4458  4096  3983  3625  
Province 1 ( 3 cities):  
  (find max, imax = 9440 0 )  
    City 0 ( 10 students): 9440  1296  283   2374  680   7426  9201  42   6348  8241  
    City 1 ( 8 students): 7513  4531  2338  6758  3213  5948  6855  2050  
    City 2 ( 6 students): 8652  4883  8499  4909  1887  1991  
Province 2 ( 3 cities):  
  (find max, imax = 9663 1 )  
    City 0 ( 8 students): 9663  5058  2432  8114  6803  2141  1821  4031  
    City 1 ( 10 students): 8480  2003  2988  4126  1733  5116  474   1249  3302  371  
    City 2 ( 6 students): 9565  4579  1766  3057  2316  9218  
Province 3 ( 3 cities):  
  (find max, imax = 9567 2 )  
    City 0 ( 10 students): 9567  5327  1808  2339  317   7691  1393  839   5040  9392  
    City 1 ( 5 students): 7392  4633  3582  4151  3797  
    City 2 ( 5 students): 8558  3763  7705  6308  1051  
Province 4 ( 3 cities):  
  (find max, imax = 9572 2 )  
    City 0 ( 6 students): 9572  3993  7296  988   8691  6233  
    City 1 ( 7 students): 8227  1517  5850  5785  2737  7258  6716
```

补充作业

- 已知某个国家，有**5个省**；每个省有随机**3-5个城市**；每个城市有**5-10名学生**；每个学生实力随机（实力值**0-10000**）。请编程模拟一个全国竞赛的过程：
 - **Step 4.** 全国比赛，第一名最牛的省排到第一位。打印结果：

```
C:\Users\pcfly\Desktop\教学 高级算法与编程\Code\Debug\Chap8_pointer.exe"
WHOLE COUNTRY COMPETITION:
-----
Province 0 wins! score = 9784!!!
-----
Now the rank is:
Province 0 ( 5 cities)g:
    City 0 ( 6 students): 9784 9296 4622 2208 2682 627
    City 1 ( 7 students): 9708 4002 5474 5447 4903 2056 6478
    City 2 ( 10 students): 9409 5341 1000 513 7154 4106 8387 7939 3365 843
    City 3 ( 7 students): 6771 2740 1079 4389 2601 2724 1843
    City 4 ( 9 students): 7807 3717 1151 2528 5023 4458 4096 3983 3625
Province 1 ( 3 cities)g:
    City 0 ( 10 students): 9440 1296 283 2374 680 7426 9201 42 6348 8241
    City 1 ( 8 students): 7513 4531 2338 6758 3213 5948 6855 2050
    City 2 ( 6 students): 8652 4883 8499 4909 1887 1991
Province 2 ( 3 cities)g:
    City 0 ( 8 students): 9663 5058 2432 8114 6803 2141 1821 4031
    City 1 ( 10 students): 8480 2003 2988 4126 1733 5116 474 1249 3302 371
    City 2 ( 6 students): 9565 4579 1766 3057 2316 9218
Province 3 ( 3 cities)g:
    City 0 ( 10 students): 9567 5327 1808 2339 317 7691 1393 839 5040 9392
    City 1 ( 5 students): 7392 4633 3582 4151 3797
    City 2 ( 5 students): 8558 3763 7705 6308 1051
Province 4 ( 3 cities)g:
    City 0 ( 6 students): 9572 3993 7296 988 8691 6233
    City 1 ( 7 students): 8227 1517 5850 5785 2737 7258 6716
    City 2 ( 7 students): 6366 2740 2403 5911 5601 5979 500
PRESS TO SEE STATISTICS...
```

补充作业

- 已知某个国家，有**5**个省；每个省有随机**3-5**个城市；每个城市有**5-10**名学生；每个学生实力随机（实力值**0-10000**）。请编程模拟一个全国竞赛的过程：
 - **Step 5.** 打印每个省的人数、第一名成绩。统计上，人数多的省往往最高分也高。特别判断第一名的省是否恰好是学生人数最多的省（仗着人多取胜）。

```
选择"C:\Users\pcfly\Desktop\教学 高级算法与编程\Code\Debug\Chap8_pointer.exe" - □ X

!-----  
CHECK NUMBER OF STUDENTS IN EACH PROVINCE && MAXIMAL SCORE:  
!-----  
  
Province 0: 5 cities, 30 students, maxscore = 9784.  
Province 1: 3 cities, 30 students, maxscore = 9440.  
Province 2: 3 cities, 24 students, maxscore = 9663.  
Province 3: 3 cities, 30 students, maxscore = 9567.  
Province 4: 3 cities, 18 students, maxscore = 9572.  
Province having the maximal #-student ( 0, 30 students) == Province having the maximal score ( 0, score=9572)  
!-----  
  
PRESS TO START 10000000 SIMULATINOS...
```

补充作业

- 已知某个国家，有**5**个省；每个省有随机**3-5**个城市；每个城市有**5-10**名学生；每个学生实力随机（实力值**0-10000**）。请编程模拟一个全国竞赛的过程：
 - **Step 6.** 请将 `malloc` 分配的内存释放（需要用循环，从最底层 `int` 数组开始释放，再释放 `int *`, `int **` 数组），结束一次模拟。

补充作业

- 已知某个国家有**5**个省，每个省随机**3-5**城市，每个城市随机**5-10**学生，实力随机（实力值**0-10000**）。请编程模拟一个全国竞赛的过程。
 - 城市为指向整型的指针；省为指向指针的指针；国家为指向指向指针的指针。模拟 **1千万** 次，看多大概率人口最多的省拿第一。
 - 注意，开辟的内存要逐级释放，如果出现问题程序会很快崩溃

A screenshot of a terminal window titled "选择"C:\Users\pcfly\Desktop\教学 高级算法与编程\Code\Debug\Chap8_pointer.exe"".

The window displays the output of a program named Chap8_pointer.exe, which performs 100,000,000 simulations. The output shows the progress of these simulations, starting from 76,000,000 and ending at 100,000,000, with a percentage completion of 100.00%.

```
Finishing 7600000 ( 76.00%) simulations...
Finishing 7800000 ( 78.00%) simulations...
Finishing 8000000 ( 80.00%) simulations...
Finishing 8200000 ( 82.00%) simulations...
Finishing 8400000 ( 84.00%) simulations...
Finishing 8600000 ( 86.00%) simulations...
Finishing 8800000 ( 88.00%) simulations...
Finishing 9000000 ( 90.00%) simulations...
Finishing 9200000 ( 92.00%) simulations...
Finishing 9400000 ( 94.00%) simulations...
Finishing 9600000 ( 96.00%) simulations...
Finishing 9800000 ( 98.00%) simulations...
Finishing 10000000 (100.00%) simulations...

In 10000000 simulations, 2944217 (29.4422%) equal.
Press any key to continue.
```

补充作业

- 已知某个国家，有**5**个省；每个省有随机**3-5**个城市；每个城市有**5-10**名学生；每个学生实力随机（实力值**0-10000**）。请编程模拟一个全国竞赛的过程：
 - Step 7.**将以上模拟进行 **1千万** 次，看有多大概率，人口最多的省能拿到第一名。
 - 注意！如果上步中**free**有问题，程序会很快用完内存而崩溃。特别，如果声明 **country[i] = (int **) malloc (sizeof(int *)*ncity);** 之后不要改变 **country[i]** 的值，否则 **free(country[i])** 会出问题。

选择"C:\Users\pcfly\Desktop\教学 高级算法与编程\Code\Debug\Chap8_pointer.exe"

```
Finishing    7600000 ( 76.00%) simulations...
Finishing    7800000 ( 78.00%) simulations...
Finishing    8000000 ( 80.00%) simulations...
Finishing    8200000 ( 82.00%) simulations...
Finishing    8400000 ( 84.00%) simulations...
Finishing    8600000 ( 86.00%) simulations...
Finishing    8800000 ( 88.00%) simulations...
Finishing    9000000 ( 90.00%) simulations...
Finishing    9200000 ( 92.00%) simulations...
Finishing    9400000 ( 94.00%) simulations...
Finishing    9600000 ( 96.00%) simulations...
Finishing    9800000 ( 98.00%) simulations...
Finishing   10000000 (100.00%) simulations...
```

In 10000000 simulations, 2944217 (29.4422%) equal.
Press any key to continue...

第十一章

结构体与共用体

1 本章要点

结构体的概念
结构体的定义和引用
结构体数组

1 主要内容

11.1 概述

11.2 定义结构体类型变量的方法

11.3 结构体变量的引用

11.4 结构体变量的初始化

11.5 结构体数组

11.6 指向结构体类型数据的指针

11.7 用指针处理链表

11.8 共用体

11.9 枚举类型

11.10 用typedef定义类型

§11.1 概述

n 问题定义：

有时需要将不同类型的数据组合成一个有机的整体，如：
一个学生有学号/姓名/性别/年龄/地址等属性

```
int num; char name[20];      char sex;  
int age; int char addr[30];
```

应当把它们组织成一个组合项，在一个组合项中包含若干个类型不同的数据项。

Num	name	sex	age	score	addr
100101	Li Fun	M	18	87.5	Beijing

图 11-1

§11.1 概述

n 声明一个结构体类型的一般形式为：

struct 结构体名
{ 成员表列 } ;

如： **struct** student

{

int num; **char** name[20]; **char** sex;

int age; **float** score; **char** addr[30];

}

结构体名

类型名

成员名

§11.2 定义结构体类型变量的方法

(1) 先声明结构体类型再定义变量名

例如 **struct** student

{

int num;**char** name[20];**char** sex;
 int age;**float** score;**char** addr[30];

};

struct student1, student2;

在定义了结构体变量后，系统会为之分配内存单元。

例如:student1和student2在内存中各占59个字节（
 $2+20+1+2+4+30=59$ ）。

§11.2 定义结构体类型变量的方法

(2) 在声明类型的同时定义变量

这种形式的定义的一般形式为：

struct 结构体名

{

成员表列

}

变量名表列；

§11.2 定义结构体类型变量的方法

例如：

```
struct student
{
    int num ;
    char name[20] ;
    char sex ;
    int age ;
    float score ;
    char addr[30] ;
}
```

student1,student2;

§11.2 定义结构体类型

(3) 直接定义结构体类型：

其一般形式为：

struct

{

成员表列

} 变量名表列；

即不出现结构体名。

注意：

(2) 对结构体中的成员（即“域”），可以单独使用，它的作用与地位相当于普通变量。

(3) 成员也可以是一个结构体变量。

(4) 成员名可以与程序中的变量名相同，二者不代表同一对象。

§11.3结构体变量的引用

引用结构体变量中成员的方式为

结构体变量名.成员名

student1.num表示student1变量中的num成员，即student1的num(学号)项。可以对变量的成员赋值,例如

student1.num=10010;

“.”是成员(分量)运算符,它在所有的运算符中优先级最高,因此可以把student1.num作为一个整体来看待。

§11.3结构体变量的引用

对结构体变量的成员可以像普通变量一样
进行各种运算（根据其类型决定可以进行
的运算）。

例如：

```
student2.score=student1.score;  
sum=student1.score+student2.score;  
student1.age++;    ++student2.age;
```

§11.4 结构体变量的定义与使用

例11.1 对结构体变量初始化

```
#include <stdio.h>
void main ()
{
    struct student
    {
        long int num ;
        char name[20];
        char sex ;
        char addr[20] ;
    } a={10101, " LiLin" , 'M' , " 123 Beijing Road" }
; /* 对结构体变量a赋初值*/
printf(" No.:%ld\nname:%s\nsex:%c\naddress:%s\n" ,
a.num , a.name , a.sex , a.addr); }
```

运行结果：

No. : 10101

name : LiLin

sex : M

address : 123 Beijing
Road

§11.5 结构体数组

一个结构体变量中可以存放一组数据（如一个学生的学号、姓名、成绩等数据）。如果有 10 个学生的数据需要参加运算，显然应该用数组，这就是结构体数组，每个数组元素都是一个结构体类型的数据。

§11.5 结构体数组

11.5.2 结构体数组的初始

与其他类型的数组一样，对结构体数组进行初始化。例如：

```
struct student
```

```
{ int num;char name[20] ; char sex; int age; float score; char address[20]; } ;stu [2] ={{10101，“Li Lin”，‘M’，18，87.5，“103 Beijing Road”}，{10102，“Zhang Fun”，‘M’，19，99，“130 Shanghai Road”}}
```



图 11-5

§11.6 指向结构体类型数据的指针

一个结构体变量的指针就是该变量所占据的内存段的起始地址。

下面通过一个简单例子来说明指向结构体变量的指针变量的应用。

例 1.1 · 3 指向结构体变量的指针的应用

```
#include <string.h>
#include <stdio.h>
void main()
{struct student{long num;
               char sex;
               float score;
}
    struct student stu_1;
    struct student* p; p=&stu_1;
    stu_1.num=89101;strcpy(stu_1.name,"LiLin");
    stu_1.sex='M';stu_1.score=89.5;
    printf(" No.:%ld\nname:%s sex:%c score:%.2f\n",
           stu_1.num, stu_1.name, stu_1.sex, stu_1.score);
    printf(" No.:%ld\nname:%s sex:%c score:%.2f\n",
           (*p).num, (*p).name, (*p).sex, (*p).score);
}
```

定义指针变量
p，指向struct
student 类型的
数据

p 指向的结
构体变量中
的成员

运行结果：

No. : 89101

name : LiLin

sex : M

score :

89.500000

No. : 89101

name : LiLin

sex : M

score : (%p).num ,

89.500000

§11.6 指向结构体类型数据的指针

以下3种形式等价：

- ① 结构体变量 · 成员名
- ② ($*p$) · 成员名
- ③ $p->$ 成员名

其中 $->$ 称为指向运算符。

- $p->n$ 得到 p 指向的结构体变量中的成员 n 的值。
◦

§ 11.7 用指针处理链表

11.7.1 链表概述

链表是一种常见的重要的数据结构,是动态地进行存储分配的一种结构。

链表的组成：

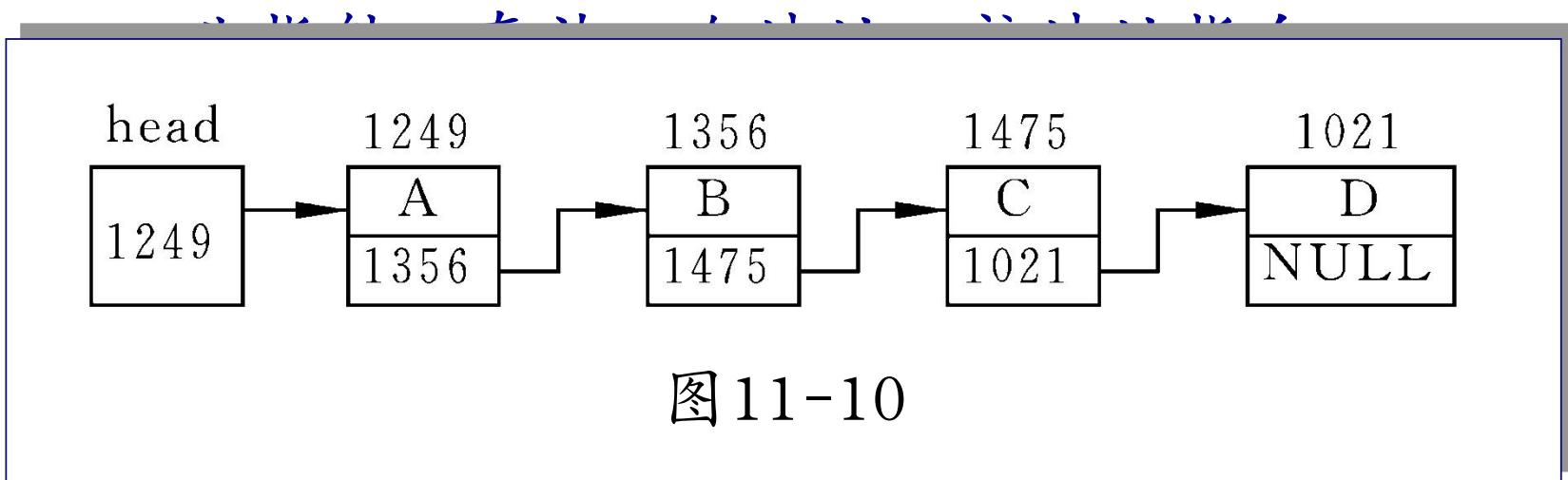


图 11-10

§ 11.7 用指针处理链表

用结构体建立链表：

```
struct student  
{    int num ;  
    float score ;
```

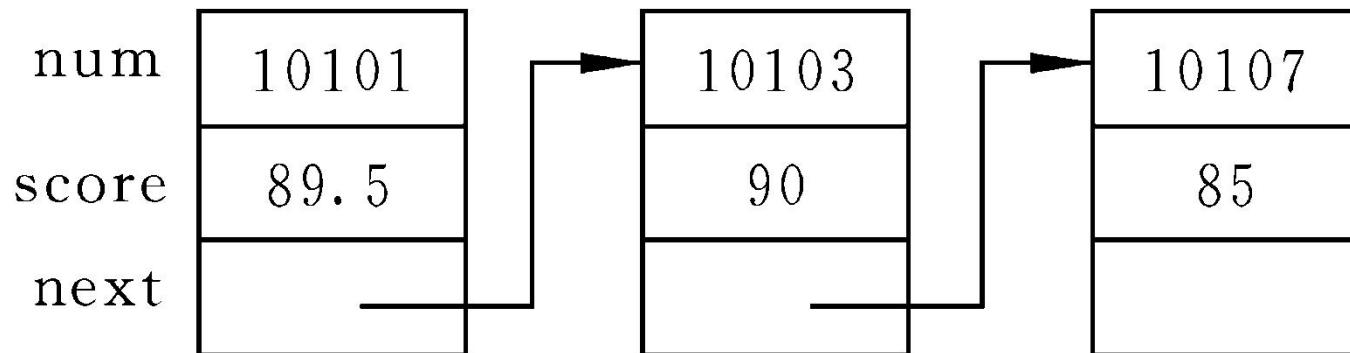


图 11-11

§ 11.7 用指针处理链表

11.7.2 简单链表

```
#include <stdio.h>
#define NULL 0
struct student
{long num;      float score; struct student *next; };
main()
{ struct student a,b,c,*head,*p;
  a. num=99101; a.score=89.5;
  b. num=99103; b.score=90;
  c. num=99107; c.score=85;
  head=&a;   a.next=&b;      b.next=&c;
  c.next=NULL;  p=head;
  do       {printf("%ld %5.1f\n",p->num,p->score);
            p=p->next;           } while(p!=NULL);
}
```

运行结果：

1010189.5
1010390.0
1010785.0

§ 11.7 用指针处理链表

程序分析：

开始时使head指向a结点，a.next指向b结点，b.next指向c结点，这就构成链表关系。

“c.next=NULL” 使c.next不指向任何有用的存储单元。

在输出链表时要借助p，先使p指向a结点，然后输出a结点中的数据，“p=p->next” 是为输出下一个结点作准备。p->next的值是b结点的地址，因此执行“p=p->next” 后p就指向b结点，所以在下一次循环时输出的是b结点中的数据。

§ 11.7 用指针处理链表（下面讲动态链表）

11.7.3 处理动态链表所需的函数

库函数提供动态地开辟和释放存储单元的有关函数：

- (1) malloc函数。
- (2) free函数
- (3) calloc函数

其函数原型为void *calloc (unsigned n ,
unsigned size) ;其作用是在内存的动态存储区
中分配n个长度为size的连续空间。函数返回
一个指向分配域起始地址的指针；如果分配不
成功，返回NULL。

§ 11.7 用指针处理链表

11.7.4 建立动态链表

在程序执行过程中从无到有地建立起一个链表，
一个一个地开辟结点和输入各结点数据，并建立起前后相链的关系

§ 11.7 用指针处理链表

例11.5 写一函数建立一个有3名学生数据的单向动态链表。

算法的实现：

我们约定学号不会为零，如果输入的学号为0，则表示建立链表的过程完成，该结点不应连接到链表中。

如果输入的 $p1 \rightarrow num$ 不等于0，则开辟新空间存放 $p1$ 结点，当前最后元素指向 $p1$ ，并把 $p1$ 作为最后元素。否则，当前链表最后元素指向NULL，程序结束。返回头结点的指针作为链表的开始。

§ 11.7 用指针处理链表

建立链表的函数如下：

```
#include <stdio.h>
#include <malloc.h>
#define NULL 0 //令NULL代表0，用它表示“空地址
#define LEN sizeof(struct student) //令LEN代表struct
//student类型数据的长度
struct student
{
    long num;
    float score;    struct student *next;
};int n; //n为全局变量，本文件模块中各函数均可使用它
```

§ 11.7 用指针处理链表

```
struct student *creat()
{struct student *head;    struct student *p1,*p2;
 n=0;
 p1=p2=( struct student*) malloc(LEN);
 scanf("%ld,%f",&p1->num,&p1->score);
 head=NULL;
 while(p1->num!=0)
 {  n=n+1;      if(n==1)head=p1;      else p2-
 >next=p1;
 p2=p1;      p1=(struct student*)malloc(LEN);
 scanf("%ld,%f",&p1->num,&p1->score);
 }
 p2->next=NULL;    return(head);}
// p1 存放“当前正在处理的结点”。
// p2 存放“已处理好的最后结点”。
```

§ 11.7 用指针处理链表

11.7.5 输出链表

首先要知道链表第一个结点的地址，也就是要知道head的值。

然后设一个指针变量p,先指向第一个结点，输出p所指的结点，然后使p后移一个结点，再输出，直到链表的尾结点。

§ 11.7 用指针处理链表

例 11·9 编写一个输出链表的函数print.

```
void print(struct student *head)
{struct student *p;
 printf("\nNow, These %d records are:\n",n);
 p=head;
 if(head!=NULL)
 do
 {printf("%ld %5.1f\n",p->num,p->score);
 p=p->next;
 }while(p!=NULL);
}
```

§ 11.7 用指针处理链表

11.7.6 对链表的删除操作

从一个动态链表中删去一个结点。

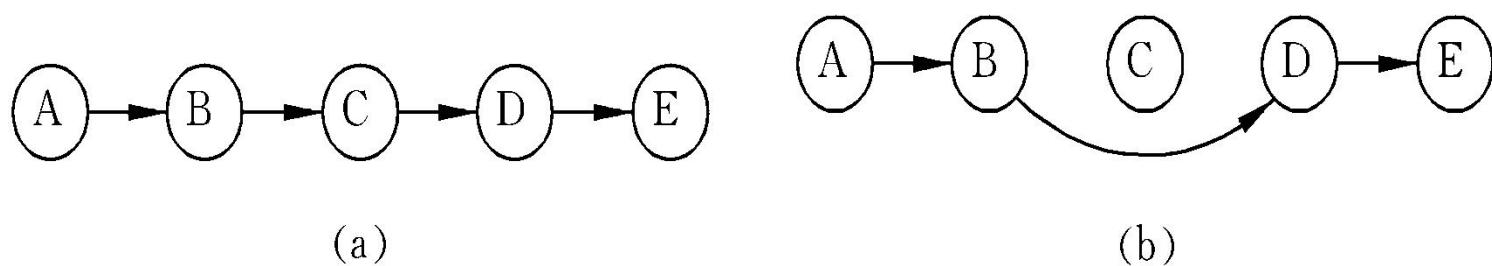


图 11-19

§ 11.7 用指针处理链表

例11.10写一函数以删除动态链表中指定的结点。

n 解题思路：

从p指向的第一个结点开始，检查该结点中的num值是否等于输入的要求删除的那个学号。如果相等就将该结点删除，如不相等，就将p后移一个结点，再如此进行下去，直到遇到表尾为止。

程序略。注意删除时应当把开辟的内存也释放。

§ 11.7 用指针处理链表

11.7.7 对链表的插入操作

对链表的插入是指将一个结点插入到一个已有的链表中。

为了能做到正确插入，必须解决两个问题：

- ① 怎样找到插入的位置；
- ② 怎样实现插入。

§ 11.7 用指针处理链表

先用指针变量p0指向待插入的结点，p1指向第一个结点

将p0->num与p1->num相比较，如果p0->num > p1-> num，则待插入的结点不应插在p1所指的结点之前。此时将p1后移，并使p2指向刚才p1所指的结点。

程序省略。

§ 11.8 共用体

11.8.1 共用体的概念

使几个不同的变量共占同一段内存的结构称为“共用体”类型的结构。

定义共用体类型变量的一般形式为：

union 共用体名

{

成员表列

} 变量表列；

1000地址

整型 i	变量
---------	----

字符变
量ch

实	型变	量	f
---	----	---	---

§ 11.8 共用体

例如：

union data

```
{ int i ;  
    char ch ; 或  
    float f ;  
} a,b,c;  
data a,b,c;
```

union data

```
{ int i ;  
    char ch ;  
    float f ;  
};union
```

§ 11.8 共用体

共用体和结构体的比较：

- 结构体变量所占内存长度是各成员占的内存长度之和。每个成员分别占有其自己的内存单元。
- 共用体变量所占的内存长度等于最长的成员的长度。

例如：上面定义的“共用体”变量 a、b、c 各占 4 个字节（因为一个实型变量占 4 个字节），而不是各占 $2 + 1 + 4 = 7$ 个字节。

§ 11.8 共用体

11.8.3 共用体类型数据的特点

- (1) 同一个内存段可以用来存放几种不同类型的成员，但在每一瞬时只能存放其中一种，而不是同时存放几种。
- (2) 共用体变量的地址和它的各成员的地址都是同一地址。
- (3) union 常用于实现以不同的名字引用结构中的元素，如：

```
union xyz{ float data[3];  
struct {float x; float y; float z;}; }
```

§13.9 枚举类型

枚举：将变量的值一一列举出来，变量的值只限于列举出来的值的范围内。

申明枚举类型用enum

enum weekday{sun , m

枚举元素
枚举常量

, tue , wed , thu , fri , sat};

定义变量：

enum weekday workday , week-day;

enum{sun , mon , tue , wed , thu , fri , sat } workday

变量值只能是sun到sat之一

§13.9 枚举类型

说明：

- (1) 在 C 编译中，对枚举元素按常量处理，故称枚举常量。它们不是变量，不能对它们赋值。
- (2) 枚举元素作为常量，它们是有值的，C 语言编译按定义时的顺序，自动给它们赋值为 0, 1, 2 ...
- (3) 枚举值可以用来作判断比较。
- (4) 一个整数不能直接赋给一个枚举变量。

§13.10 用typedef定义类型

用typedef声明新的类型名来代替已有的类型名

声明INTEGER为整型

```
typedef int INTEGER
```

声明结构类型

```
typedef struct{  
    int month ;  
    int day ;  
    int year ; }DATE;
```

§13.10 用typedef定义类型

声明NUM为整型数组类型

```
typedef int NUM[100];
```

声明STRING为字符指针类型

```
typedef char *STRING;
```

声明POINTER为指向函数的指针类型，该函数返回
整型值

```
typedef int (*POINTER)()
```

§13.10 用typedef定义类型

用typedef定义类型的方法

- ① 先按定义变量的方法写出定义体（如：int i）。
- ② 将变量名换成新类型名（例如：将i换成COUNT）。
- ③ 在最前面加 `typedef`
(例如：`typedef int COUNT`)。
- ④ 然后可以用新类型名去定义变量。

§13.10 用typedef定义类型

用typedef定义类型的方法（举例）

- ① 先按定义数组变量形式书写：int n[100]；
- ② 将变量名n换成自己指定的类型名：

```
int NUM [ 1 0 0 ] ;
```

- ③ 在前面加上typedef，得到

```
typedef int NUM [ 1 0 0 ] ;
```

- ④ 用来定义变量：NUM n；

§13.10 用typedef定义类型

说明：

- (1) 用typedef可以声明各种类型名，但不能用来定义变量。
- (2) 用typedef只是对已经存在的类型增加一个类型名，而没有创造新的类型。
- (3) 当不同源文件中用到同一类型数据时，常用typedef声明一些数据类型，把它们单独放在一个文件中，然后在需要用到它们的文件中用#include命令把它们包含进来。
- (4) 使用typedef有利于程序的通用与移植。

§13.10 用typedef定义类型

说明：

(5) `typedef`与`#define`有相似之处，例如：

`typedef int COUNT;`与`#define COUNT int`的作用都是用`COUNT`代表`int`。但事实上，它们二者是不同的。`#define`是在预编译时处理的，它只能作简单的字符串替换，而`typedef`是在编译时处理的。实际上它并不是作简单的字符串替换，而是采用如同定义变量的方法那样来声明一个类型

本周作业

- * 课后题 5 6 7 8 9 10。不分单复。键盘输入都可以用随机数
- * 自己学习下共用体、枚举、`typedef`，各写一个程序练习。
- * 自学位运算，完成两道课后题。
- * 准备好python IDE：

在自己笔记本安装 Anaconda3 。打开 Jupyter notebook
(windows下：左下角输入框输入 Jupyter ；
Linux 下：命令行 `ipython notebook`)，
确保能正常工作 (输入 `1+1`，`shift+enter` 运行得2)，下次
上课时要用。

第十二章

位运算

1 主要内容

12.1位运算符和位运算

12.2位运算举例

12.3位段

概念

- 1 位运算是指按二进制位进行的运算。因为在系统软件中，常要处理二进制位的问题。
- 1 例如：将一个存储单元中的各二进制位左移或右移一位，两个数按位相加等。
- 1 C语言提供位运算的功能，与其他高级语言（如PASCAL）相比，具有很大的优越性。

§12.1 位运算符和位运算

C语言提供的位运算符有：

运算符	含义	运算符	含义
&	按位与	~	取反
	按位或	<<	左移
^	按位异或	>>	右移

说明：

- (1)除～以外，均为二目（元）运算符，即要求两侧各有一个运算量。
- (2)运算量只能是整型、字符型的数据，不能为实型数据。

12.1.1“按位与”运算符 (&)

按位与是指：参加运算的两个数据，按二进制位进行“与”运算。如果两个相应的二进制位都为 1，则该位的结果值为 1；否则为 0。即：

$$0 \& 0 = 0, 0 \& 1 = 0, 1 \& 0 = 0, 1 \& 1 = 1$$

例：3 & 5 并不等于 8，应该是按位与运算：

00000011(3)

& 00000101(5)

—————
00000001(1)

3&5的值得

1

注意：如果参加&运算是负数（如 -3&-5），则要以补码形式表示为二进制数，然后再按位进行“与”运算。

补码的概念

1. 正整数的补码是其二进制表示，与原码相同。

【例1】+9的补码是00001001。

2. 求负整数的补码，将其对应正数二进制表示所有位取反（包括符号位，0变1，1变0）后加1

注意：最左边的一位是符号位。零代表正，1代表负。

【例2】求-5的补码。

-5对应正数5（00000101）

→所有位取反（11111010）

→加1(11111011)

所以-5的补码是11111011。

补码的概念

【例3】数0的补码表示是唯一的。

正零和负零的补码一样。

$$[+0]_{\text{补}} = [+0]_{\text{原}} = 00000000$$

$$[-0]_{\text{补}} = 11111111 + 1 = 00000000$$

转化为原码

补码求原码的操作就是对该补码再求补码：

(1)如果补码的符号位为“0”，表示是一个正数，其原码就是补码。

(2)如果补码的符号位为“1”，表示是一个负数，那么求给定的这个补码的补码就是要求的原码。

【例4】已知一个补码为11111001，则原码是
00000111 (-7)。

取反后为0000110；再加1，所以是00000111。
反之， 00000111取反为11111000，再加1得到
11111001。

补码与原码相互转换，其运算过程是相同的，不需要额外的硬件电路。是使用补码的一个优点。

按位与的用途：

- (1) 清零
- (2) 取一个数中某些指定位

如有一个整数 a (2个字节)，想要取其中的低字节，只需将 a 与8个1按位与即可。

a	00101100	10101100
b	00000000	11111111
c	00000000	10101100

12.1.2 “按位或” 运算符 ($|$)

两个相应的二进制位中只要有一个为 1，该位的结果值为 1。

即 $0|0=0$ ， $0|1=1$ ， $1|0=1$ ， $1|1=1$

例： 060|017, 将八进制数60与八进制数17进行按位或运算。

$$\begin{array}{r} 00110000 \\ | \quad 00001111 \\ \hline 00111111 \end{array}$$

应用：按位或运算常用来对一个数据的某些位定值为 1。

例如：如果想使一个数 a 的低 4 位改为 1，只需将 a 与 017 进行按位或运算即可。

12.1.3“异或”运算符 (\wedge)

异或运算符 \wedge 也称 XOR 运算符。它的规则是：
若参加运算的两个二进制位同号则结果为 0 (假)
异号则结果为 1 (真)

即: $0 \wedge 0 = 0$, $0 \wedge 1 = 1$, $1 \wedge 0 = 1$, $1 \wedge 1 = 0$

例 :

$$\begin{array}{r} 00111001 \\ \wedge 00101010 \\ \hline 00010011 \end{array}$$

即: $071 \wedge 052 = 023$ (八进制数)

^运算符应用：

(1) 使特定位翻转

设有01111010，想使其低4位翻转，即1变为0，0变为1。可以将它与00001111进行^运算，即：

运算结果的低4位正好是原数低4位的翻转。可见，要使哪几位翻转就将与其进行^运算的该几位置为1即可。


$$\begin{array}{r} 01111010 \\ \wedge \quad 00001111 \\ \hline 01110101 \end{array}$$

(2) 交换两个值，不用临时变量

例如： $a = 3$ ， $b = 4$ 。

想将 a 和 b 的值互换，可以用以下赋值语句实现：

$a = a \wedge b;$

$b = a \wedge b;$

$a = a \wedge b;$ $a = 0\ 1\ 1$
 $(\wedge)\ b = 1\ 0\ 0$

$a = 1\ 1\ 1$ ($a \wedge b$ 的结果， a 已变成 7)

$(\wedge)\ b = 1\ 0\ 0$

$b = 0\ 1\ 1$ ($b \wedge a$ 的结果， b 已变成 3)

$(\wedge)\ a = 1\ 1\ 1$

$a = 1\ 0\ 0$ ($a \wedge b$ 的结果， a 已变成 4)

12.1.4 “取反” 运算符 (\sim)

\sim 是一个单目（元）运算符，用来对一个二进制数按位取反，即将0变1，将1变0。

例如， ~ 025 是对八进制数25（即二进制数00010101）按位求反。

000000000010101

(\sim) 帽

111111111101010 (八进制数177752)

12.1.5 左移运算符 (<<)

左移运算符是用来将一个数的各二进制位全部左移若干位。

例如： $a=a<<2$ 将 a 的二进制数左移 2 位，右补 0。

若 $a = 15$ ，即二进制数 00001111，
左移 2 位得 0011100，(十进制数 60)

高位左移后溢出，舍弃。

12.1.5 左移运算符（<<）

左移 1 位相当于该数乘以 2，左移 2 位相当于该数乘以 $2^2 = 4$, $15 << 2 = 60$ ，即乘了 4。但此结论只适用于该数左移时被溢出舍弃的高位中不包含 1 的情况。

12.1.6 右移运算符(>>)

右移运算符是 $a>>2$ 表示将a的各二进制位右移2位，移到右端的低位被舍弃,对无符号数,高位补0。

例如： a=017时：

a的值用二进制形式表示为00001111，
舍弃低2位11： $a>>2=00000011$

右移一位相当于除以2

右移n位相当于除以 2^n 。

在右移时,需要注意符号位问题:

对于有符号的值,如果符号位原来为1(即负数),则左边移入0还是1,要取决于所用的计算机系统。

有的系统移入0,有的系统移入1。移入0的称为“逻辑右移”,即简单右移;移入1的称为“算术右移”。

例： a的值是八进制数113755：

a:1001011111101101 (用二进制形式表示)

a>>1: 010010111110110 (逻辑右移时)

a>>1: 110010111110110 (算术右移时)

在有些系统中,a>>1得八进制数045766,而在另一些系统上可能得到的是145766。

我在自己电脑上测试，发现我的 Visual C 是
算术右移。

12.1.7 位运算赋值运算符

位运算符与赋值运算符可以组成复合赋值运算符。

例如: $\&=$, $|=$, $>>=$, $<<=$, $\wedge=$

例 : $a \&= b$ 相当于 $a = a \& b$

$a <<= 2$ 相当于 $a = a << 2$

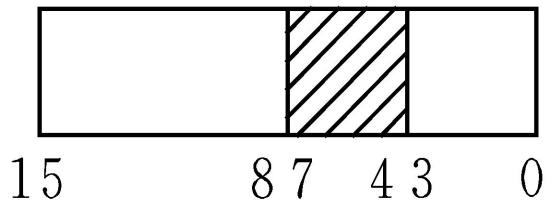
$<< 2$

§12.2 位运算举例

例12.1 取一个整数a从右端开始的4~7位

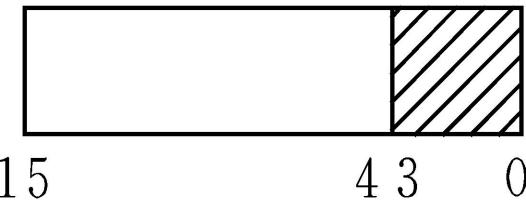
① 先使a右移4位： $a >> 4$

目的是使要取出的那几位移到最右端



(a)

未右移时的情况



(b)

右移4位后的情况

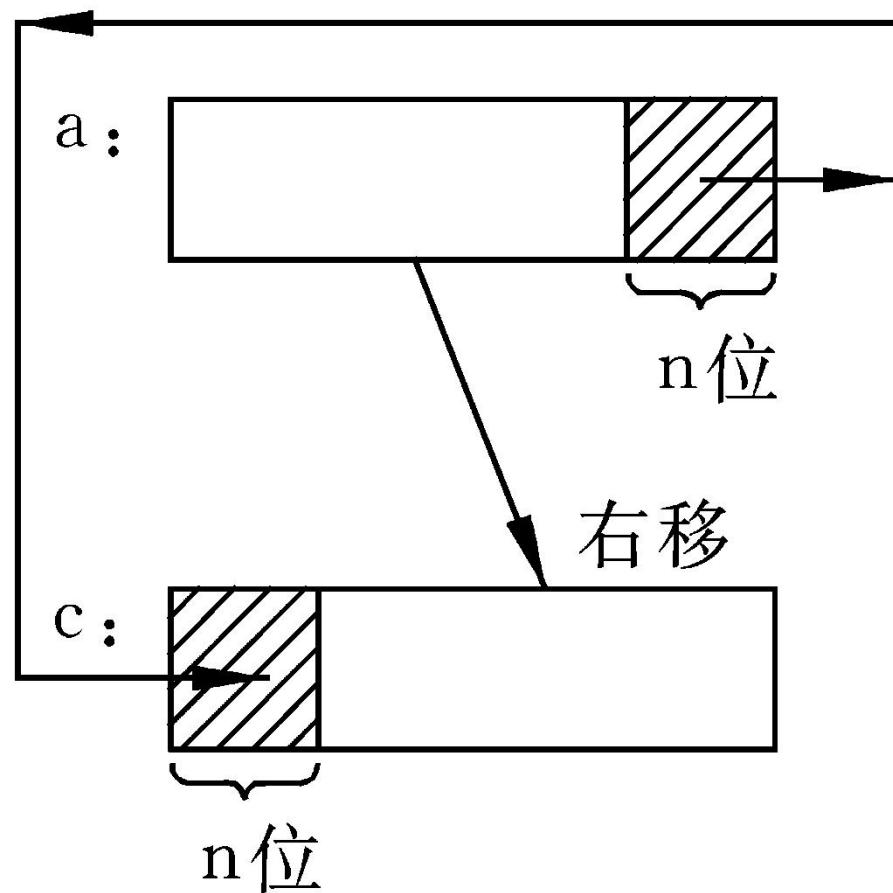
② 设置一个低4位全为1,其余全为0的数 $\sim (\sim 0 << 4)$

③ 将上面①、②进行&运算： $(a >> 4) \& \sim (\sim 0 << 4)$

例12.2 循环移位

要求将 a 进行右循环移位

将 a 右循环移 n 位，即将 a 中原来左面 ($16 - n$) 位右移 n 位，原来右端 n 位移到最左面 n 位。



步骤：

- ① 将 a 的右端 n 位先放到 b 中的高 n 位中，
实现语句： $b = a << (16 - n)$ ；
- ② 将 a 右移 n 位，其左面高位 n 位补 0，
实现语句： $c = a >> n$ ；
- ③ 将 c 与 b 进行按位或运算，即 $c = c | b$ ；

§12.3 位段

信息的存取一般以字节为单位。实际上，有时存储一个信息不必用一个或多个字节，例如，“真”或“假”用 0 或 1 表示，只需 1 位即可。

在计算机用于过程控制、参数检测或数据通信领域时，控制信息往往只占一个字节中的一个或几个二进制位，常常在一个字节中放几个信息。

怎样向一个字节中的一个或几个二进制位赋值和改变它的值呢？可以用以下两种方法：

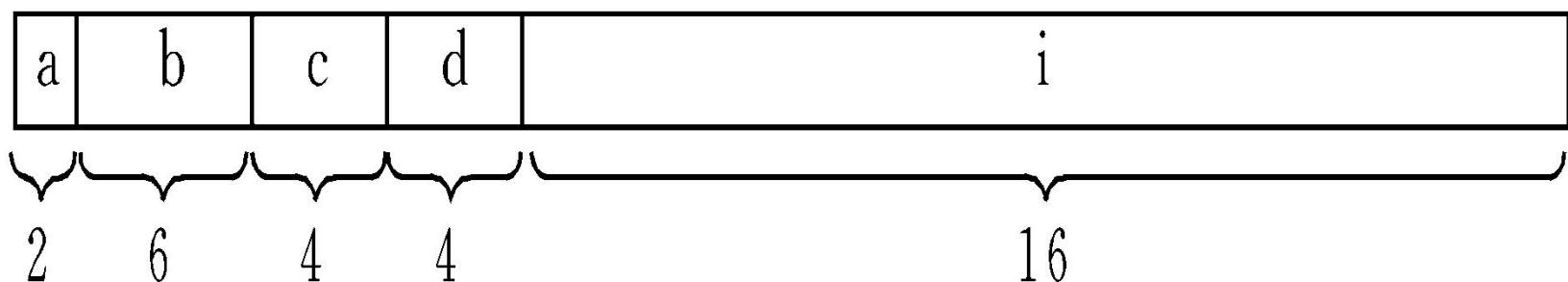
- (1) 可以人为地将一个整型变量data分为几部分。
但是用这种方法给一个字节中某几位赋值太麻烦。可以位段结构体的方法。

(2) 位段

C语言允许在一个结构体中以位为单位来指定其成员所占内存长度，这种以位为单位的成员称为“位段”或称“位域”(bit field)。利用位段能够用较少的位数存储数据。

程序如下：

```
struct packed-data
{ unsigned a : 2 ;
  unsigned b : 6 ;
  unsigned c : 4 ;
  unsigned d : 4 ;
  int i ;
} data ;
```



关于位段的定义和引用的说明：

- (1) 位段成员的类型必须指定为unsigned或int类型。
- (2) 若某一位段要从另一个字开始存放，可用以下形式定义：

unsigned a : 1 ;

unsigned b : 2 ; 一个存储单元

unsigned : 0 ;

unsigned c : 3 ; 另一存储单元

a、b、c应连续存放在一个存储单元中，这里用了长度为0的位段，其作用是使下一个位段从下一个存储单元开始存放。因此，只将a、b存储在一个存储单元中，c另存在下一个单元(“存储单元”可能是一个字节，也可能是2个字节，取决于编译系统)。

关于位段的定义和引用的说明：

- (3) 一个位段必须存储在同一存储单元中，不能跨两个单元。如果第一个单元空间不能容纳下一个位段，则该空间不用，而从下一个单元起存放该位段。
- (4) 可以定义无名位段。
- (5) 位段的长度不能大于存储单元的长度，也不能定义位段数组。
- (6) 位段可以用整型格式符输出。
- (7) 位段可以在数值表达式中引用，它会被系统自动地转换成整型数。

练习题

编写一个程序，通过命令行参数读取两个二进制字符串，对这两个二进制数使用`~`运算符、`&`运算符、`|`运算符和`^`运算符，并以二进制字符串形式打印结果（如果无法使用命令行环境，可以通过交互式让程序读取字符串）。

编写一个程序，接受两个int类型的参数：一个是值；一个是位的位置。如果指定位的位置为1，该函数返回1；否则返回0。在一个程序中测试该函数。

第十三章

文 件

1

本章要点

- n 文件的基本概念
- n 文件的基本函数
- n 文件的顺序读写
- n 文件的随机读写
- n 文件简单应用

1 主要内容

13.1 C 文件概述

13.2 文件类型指针

13.3 文件的打开与关闭

13.4 文件的读写

13.5 文件的定位

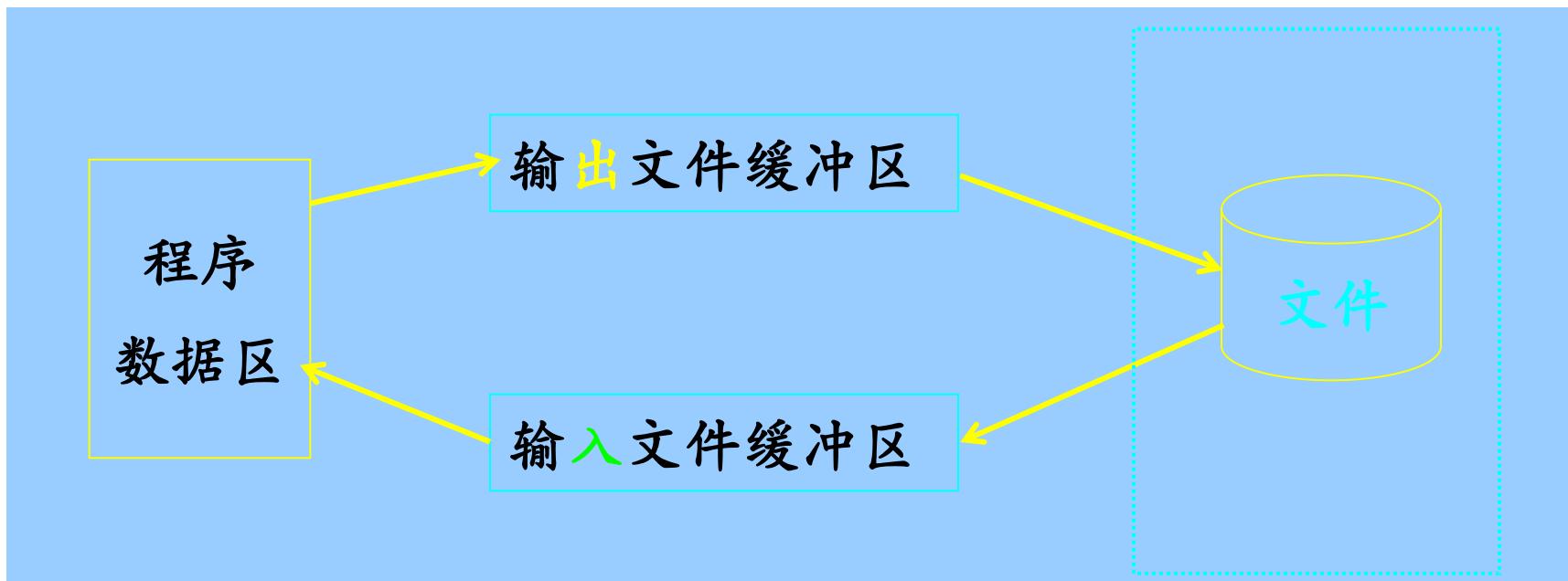
13.6 出错的检测

13.7 文件输入输出小结

§13.1 C文件概述

文件的定义

文件指存储在外部介质(如磁盘磁带)上数据的集合.
操作系统是以文件为单位对数据进行管理的.



§13.1 C文件概述(续)

文件的分类

- 从用户观点：

特殊文件(标准输入输出文件或标准设备文件)

普通文件(磁盘文件)

从操作系统的角度看，每一个与主机相连的输入输出设备看作是一个文件。

例：输入文件：终端键盘

输出文件：显示屏和打印机

§13.1 C文件概述(续)

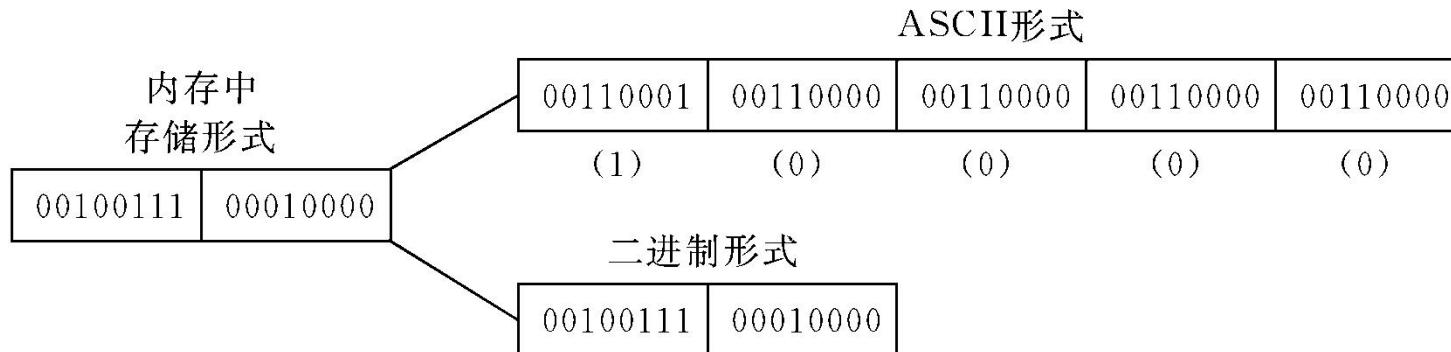
文件的分类

- 按数据的组织形式：

ASCII文件(文本文件):每一个字节放一个ASCII代码

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磁盘上存放.

例：整数10000在内存中的存储形式以及分别按ASCII码形式和二进制形式输出如下图所示：



§13.1 C文件概述(续)

文件的分类

ASCII文件和二进制文件的比较：

ASCII文件便于对字符进行逐个处理，也便于输出字符。但一般占存储空间较多，而且要花费转换时间。

二进制文件可以节省外存空间和转换时间，但一个字节并不对应一个字符，不能直接输出字符形式。

一般中间结果数据需要暂时保存在外存上，以后又需要输入内存的，常用二进制文件保存。

§13.1 C文件概述(续)

文件的分类

- C语言对文件的处理方法：

缓冲文件系统：系统自动地在内存区为每一个正在使用的文件开辟一个缓冲区。用缓冲文件系统进行的输入输出又称为**高级磁盘输入输出**。

非缓冲文件系统：系统不自动开辟确定大小的缓冲区，而由程序为每个文件设定缓冲区。用非缓冲文件系统进行的输入输出又称为**低级输入输出系统**。

§13.1 C文件概述(续)

说明:

在UNIX系统下,用缓冲文件系统来处理文本文件,
用非缓冲文件系统来处理二进制文件.

ANSI C 标准只采用缓冲文件系统来处理文本文件
和二进制文件.

C语言中对文件的读写都是用库函数来实现.

§13.2 文件类型指针

Turbo C 在 stdio.h 文件中有以下的文件类型声明：

```
typedef struct
```

```
{ short level; /*缓冲区“满”或“空”的程度*/  
  unsigned flags; /*文件状态标志*/  
  char fd; /*文件描述符*/  
  unsigned char hold; /*如无缓冲区不读取字符*/  
  short bsize; /*缓冲区的大小*/  
  unsigned char *buffer; /*数据缓冲区的位置*/  
  unsigned char *curp; /*指针，当前的指向*/  
  unsigned istemp; /*临时文件，指示器*/  
  short token; /*用于有效性检查*/ } FILE;
```

在缓冲文件系统中, 每个被使用的文件都要在内存中开辟一个 FILE 类型的区, 存放文件的有关信息.

§13.2 文件类型指针(续)

FILE类型的数组：

FILE f [5]; 定义了一个结构体数组f，它有5个元素，可以用来存放5个文件的信息。

文件型指针变量：

FILE *fp; fp是一个指向FILE类型结构体的指针变量。

可以使fp指向某一个文件的结构体变量，从而通过该结构体变量中的信息能够访问该文件。

如果有n个文件，一般应设n个指针变量，使它们分别指向n个文件，以实现对文件的访问。

§13.3 文件的打开与关闭

一.文件的打开(fopen函数)

函数调用：

FILE *fp ;

fp=fopen (文件名 , 使用文件方式) ;

- ①需要打开的文件名，也就是准备访问的文件的名字；
- ②使用文件方式（“读”、“写”... 见下一页表）；

§13.3 文件的打开与关闭(续)

文件使用方式	含 义
“r”	(只读，文本)为 输入 打开一个 文本 文件
“w”	(只写，文本)为 输出 打开一个 文本 文件
“a”	(追加，文本)向 文本 文件尾增加数据
“rb”	(只读，二进制)为 输入 打开一个 二进制 文件
“wb”	(只写，二进制)为 输出 打开一个 二进制 文件
“ab”	(追加，二进制)向 二进制 文件尾增加数据
“r+”	(读写，文本)为 读/写 打开一个 文本 文件
“w+”	(读写，文本)为 读/写建立 一个 新的文本 文件
“a+”	(读写，文本)为 读/写 打开一个 文本 文件
“rb+”	(读写，二进制)为 读/写 打开一个 二进制 文件
“wb+”	(读写，二进制)为 读/写建立 一个 新的二进制 文件
“ab+”	(读写，二进制)为 读/写 打开一个 二进制 文件

下面内容选自

<https://www.cnblogs.com/kangjianwei101/p/5220021.html>

在C语言的文件操作语法中，打开文件文件有以下12种模式，如下图：

打开模式	只可以读	只可以写	读写兼备			
文本模式	r	w	a	r+	w+	a+
二进制模式	rb	wb	ab	rb+ (r+b)	wb+ (w+b)	ab+ (a+b)

打开模式	简述	若欲操作的文件不存在	成功打开文件后文件指针位置	是否清空原有内容	读取位置	写入位置	注
r	只读	打开失败	开头	否	任意位置读取	不可写入	-
w	只写	新建	开头	是	不可读取	任意位置写入	写入时会覆盖原有位置内容
a		新建	结尾	否	不可读取	只能尾部写入	-
r+	读写	打开失败	开头	否	任意位置读取	任意位置写入	写入时会覆盖原有位置内容
w+		新建	开头	是	任意位置读取	任意位置写入	写入时会覆盖原有位置内容
a+		新建	结尾	否	任意位置读取	只能尾部写入	-

§13.3 文件的打开与关闭(续)

二.文件的关闭(fclose函数)

函数调用:

`fclose (文件指针) ;`

函数功能:

使文件指针变量不指向该文件，也就是文件指针变量与文件“脱钩”，此后不能再通过该指针对原来与其相联系的文件进行读写操作

返回值:

关闭成功返回值为 0 ；否则返回EOF(-1)

§13.4 文件的读写

一. 字符输入输出函数(fputs()和fgets())

fputs函数

函数调用:

`fputs (ch , fp) ;`

函数功能:

将字符 (ch的值) 输出到fp所指向的文件中去。

返回值:

如果输出成功，则返回值就是输出的字符；

如果输出失败，则返回一个EOF.

§13.4 文件的读写(续)

fgets函数

函数调用:

ch=fgets (fp) ;

函数功能:

从指定的文件读入一个字符,该文件必须是以读或
读写方式打开的。

返回值:

读取成功一个字符，赋给 ch。

如果遇到文件结束符，返回一个文件结束标志
EOF。

§13.4 文件的读写(续)

常见的读取字符操作

从一个**文本文件**顺序读入字符并在屏幕上显示出来：

```
ch = fgetc (fp) ;
while (ch !=EOF)
{
    putchar (ch) ;
    ch = fgetc (fp) ;
}
```

注意：EOF不是可输出字符，因此不能在屏幕上显示。由于字符的ASCII码不可能出现-1，因此EOF定义为-1是合适的。当读入的字符值等于-1时，表示读入的已不是正常的字符而是文件结束符。

§13.4 文件的读写(续)

常见的读取字符操作

从一个**二进制文件**顺序读入字符：

```
while ( !feof ( fp ) )
{
    ch = fgetc ( fp ) ;
}
```

注意：ANSI C提供一个feof（）函数来判断文件是否真的结束。如果是文件结束，函数feof（fp）的值为1（真）；否则为0（假）。以上也适用于文本文件的读取。

```
#include <stdlib.h>
#include <stdio.h>
void main(void)
{ FILE *fp;
    char ch,filename[10];
    scanf("%s",filename);
    if((fp=fopen(filename,"w"))==NULL) {
        printf("cannot open file\n");
        exit(0); /*终止程序*/
    }
    ch=getchar(); /*接收执行scanf语句时最后输入的回车符 */
    ch=getchar(); /* 接收输入的第一个字符 */
    while(ch != '\n')
```

运行情况如下：

f i l e 1 . c (输入磁盘文件名)

c o m p u t e r a n d c # (输入一个字符串)

c o m p u t e r a n d c (输出一个字符串)

```
#include <stdlib.h>
#include <stdio.h>
main()
{FILE *in,*out;
char ch,infile[10],outfile[10];
printf("Enter the infile name:\n");
scanf("%s",infile);
printf("Enter the outfile name:\n");
scanf("%s",outfile);
if((in=fopen(infile,"r"))==NULL)

```

运行情况如下：

Enter the infile name

file1.c (输入原有磁盘文件名)

Enter the outfile name:

file2.c (输入新复制的磁盘文件名)

程序运行结果是将 file1.c 文件中的内容复制到
file2.c 中去。

```
#include <stdlib.h>
#include <stdio.h>
main(int argc,char *argv[ ])
{FILE *in,*out;
char ch;
if (argc!=3)
{
    printf("You forgot to enter a filename\n");
    exit(0); }
if((in=fopen(argv[1],"rb"))==NULL)
```

运行方法：

设经编译连接后得到的可执行文件名为a.exe，则在DOS命令工作方式下，可以输入以下的命令行：

C>a file1.c file2.c

file1.c和file2.c，分别输入到argv[1]和argv[2]中，argv[0]的内容为a，argc的值等于3。

§13.4 文件的读写(续)

二. 数据块读写函数(fread()和fwrite())

函数调用：

`fread (buffer,size,count , fp);`

`fwrite(buffer,size,count,fp);`

参数说明：

buffer：是一个指针。

对**fread** 来说，它是读入数据的存放地址。

对**fwrite**来说，是要输出数据的地址（均指起始地址）。

size： 要读写的字节数。

count： 要进行读写多少个**size**字节的数据项。

fp： 文件型指针。

§13.4 文件的读写(续)

使用举例：

若文件以二进制形式打开：

`fread(f,4,2,fp);`

此函数从`fp`所指向的文件中读入2个4个字节的数据，存储到数组`f`中。

§13.4 文件的读写(续)

使用举例：

若有如下结构类型：

```
struct student_type  
{char name[10];  
int num;  
int age;  
char addr[30];}stud[40];
```

可以用fread和fwrite来进行数据的操作：

```
for ( i = 0 ; i < 40 ; i ++ )  
    fread(&stud [i] , sizeof(struct student-type) , 1 , fp);  
for ( i = 0 ; i < 40 , i ++ )  
    fwrite(&stud [i] , sizeof(struct student-type) , 1 , fp);
```

§13.4 文件的读写(续)

使用举例：

例 13·3 从键盘输入 4 个学生的有关数据，然后把它们转存到磁盘文件上去。

```
#include <stdio.h>
#define SIZE 4
struct student_type
{
    char name[10];
    int num;
    int age;
    char addr[15];
}stud[SIZE]; /*定义结构*/
```

```
m
void save( )
{FILE *fp;
 int i;
 if((fp=fopen("stu-list","wb"))==NULL)
 { printf("cannot open file\n");
 return;}
 for(i=0;i<SIZE;i++)/*二进制写*/
 if(fwrite(&stud[i],sizeof(struct student_type),1,fp)!=1)
```

运行情况如下：

输入 4 个学生的姓名、学号、年龄和地址：

Z h a n g 1 0 0 1 1 9 r o o m - 1 0 1

F u n 1 0 0 2 2 0 r o o m - 1 0 2

T a n 1 0 0 3 2 1 r o o m - 1 0 3

L i n g 1 0 0 4 2 1 r o o m - 1 0 4

```
#include <stdio.h>
#define SIZE 4
struct student_type
{
    char name[10];
    int num;
    int age;
    char addr[15];
}stud[SIZE];
main( )
```

屏幕上显示出以下信息：

Z h a n g	1 0 0 1	1 9	r o o m - 1 0 1
F u n	1 0 0 2	2 0	r o o m - 1 0 2
T a n	1 0 0 3	2 1	r o o m - 1 0 3
L i n g	1 0 0 4	2 1	r o o m - 1 0 4
.....			

§13.4 文件的读写(续)

如果已有的数据已经以二进制形式存储在一个磁盘文件“stu.dat”中，要求从其中读入数据并输出到“student”文件中，可以编写一个load函数，从磁盘文件中读二进制数据。

```
void load()
{FILE *fp;int i;
if((fp=fopen("stu-dat","rb"))==NULL)
{ printf("cannot open infile\n");
return;}
for(i=0;i<SIZE;i++)
if(fread(&stud[i],sizeof(struct student_type),1,fp)!=1)
{if(feof(fp)) {fclose(fp); return;}
printf("file read error\n");}
fclose (fp) ; }
```

§13.4 文件的读写(续)

三. 格式化读写函数(`fprintf()`和`fscanf()`)

函数调用:

`fprintf (文件指针, 格式字符串, 输出表列) ;`
`fscanf (文件指针, 格式字符串, 输入表列) ;`

函数功能:

注意:

用`fprintf`和`fscanf`函数对磁盘文件读写，使用方便，容易理解，但由于在输入时要将ASCII码转换为二进制形式，在输出时又要将二进制形式转换成字符，花费时间比较多。因此，在内存与磁盘频繁交换数据的情况下，最好不用`fprintf`和`fscanf`函数，而用`read`和`write`函数。

§13.4 文件的读写(续)

三. 其他读写函数

putw()和**getw()**

函数调用：

```
putw(int i,FILE * fp);  
int i = getw(FILE * fp);
```

函数功能：

对磁盘文件中读写一个字（整数）

例：

```
putw(10,fp);  
i = getw(fp);
```

getw函数定义如下：

```
getw(FILE *fp)  
{  
    char s;  
    s=char *&i;  
    s[0] = getc(fp);  
    s[1] = getc(fp);  
    return i;  
}
```

§13.4 文件的读写(续)

用户自定义读取其他类型数据的函数

向磁盘文件写一个实数（用二进制方式）的函数putfloat：

```
putfloat(float num,FILE *fp)
{
    char s;
    int count;
    s = (char*)&num;
    for(count = 0;count < 4;count++)
        putc(s[count],fp);
}
```

§13.4 文件的读写(续)

fgets函数

函数作用：

从指定文件读入一个字符串。

函数调用：

`fgets(str,n,fp);`

从`fp`指向的文件读入`n-1`个字符，在最后加一个'\0'

返回值：

`str`的首地址

§13.4 文件的读写(续)

fputs函数

函数作用：

向指定的文件输出一个字符串。

函数调用：

fputs(str,fp);

第一个参数可以是字符串常量、字符数组名或字符型指针。字符串末尾的'\\0'不输出。

返回值：

输入成功，返回值为0；

输入失败，返回EOF.

§13.5 文件的定位

```
#include<stdio.h>
main()
{
    FILE *fp1,*fp2;
    fp1=fopen("file1.c","r");
    fp2=fopen("file2.c","w");
    while(!feof(fp1)) putchar(getc(fp1));
    rewind(fp1);
    while(!feof(fp1))
        putc(getc(fp1),fp2);
    fclose(fp1);fclose(fp2);
}
```

§13.5 文件的定位

顺序读写和随机读写

顺序读写：

位置指针按字节位置顺序移动。

随机读写：

读写完上一个字符（字节）后，并不一定要读写其后续的字符（字节），而可以读些文件中任意位置上所需要的字符（字节）。

§13.5 文件的定位

fseek函数（一般用于二进制文件）

函数功能：

改变文件的位置指针

函数调用形式：

fseek(文件类型指针，位移量，起始点)

起始点：文件开头 SEEK_SET 0

 文件当前位置 SEEK_CUR 1

 文件末尾 SEEK_END 2

位移量：以起始点为基点，向前移动的字节数。一般
要求为long型

§13.5 文件的定位

fseek函数应用举例

`fseek(fp, 100L, 0);`

将位置指针移到离文件头100个字节处

`fseek(fp, 50L, 1);`

将位置指针移到离当前位置50个字节处

`fseek(fp, 50L, 2);`

将位置指针从文件末尾处向后退10个字节

```
#include <stdlib.h>
#include<stdio.h>
struct student_type
{
    char name[10];
    int num;
    int age;
    char sex;
}stud[10];    main()
{ int i;
FILE *fp;
if((fp=fopen("stud-dat","rb"))==NULL)
{printf("can not open file\n");
 exit(0);}
for(i=0;i<10;i+=2)
{fseek(fp,i*sizeof(struct student_type),0);
 fread(&stud[i], sizeof(struct student_type),1,fp);
printf("%s %d %d %c\n",stud[i].name,
stud[i].num,stud[i].age,stud[i].sex);}
fclose(fp)}
```

§13.5 文件的定位

ftell函数

函数作用：

得到流式文件中的当前位置，用相对于文件开头的位移量来表示。

返回值：

返回当前位置，出错时返回-1L。

应用举例：

```
i = ftell(fp);  
if(i== -1L) printf("error\n");
```

§13.6 出错的检测

ferror函数

调用形式：

`ferror(fp);`

返回值：

返回0，表示未出错；返回非0，表示出错。

在调用一个输入输出函数后立即检查ferror函数的值，否则信息会丢失。在执行fopen函数时，ferror函数的初始值自动置为0。

§13.6 出错的检测

clearerr函数

调用形式：

clearerr(fp);

函数作用：

使文件错误标志和文件结束标志置为0。

只要出现错误标志，就一直保留，直到对同一文件调用clearerr函数或rewind函数，或任何其他一个输入输出函数。

§13.7 文件输入输出小结

分类	函数名	功能
打开文件	fopen()	打开文件
关闭文件	fclose()	关闭文件
文件定位	fseek()	改变文件位置指针的位置
	rewind()	使文件位置指针重新至于文件开头
	tell()	返回文件位置指针的当前值
文件状态	feof()	若到文件末尾，函数值为真
	ferror()	若对文件操作出错，函数值为真
	clearerr()	使 ferror 和 feof() 函数值置零

§13.7 文件输入输出小结

<u>分类</u>	<u>函数名</u>	<u>功能</u>
文件读写	fgetc(),getc()	从指定文件取得一个字符
	fputc(),putc()	把字符输出到指定文件
	fgets()	从指定文件读取字符串
	fputs()	把字符串输出到指定文件
	getw()	从指定文件读取一个字（int型）
	putw()	把一个字输出到指定文件
	fread()	从指定文件中读取数据项
	fwrite()	把数据项写到指定文件中
	fscanf()	从指定文件按格式输入数据
	fprintf()	按指定格式将数据写到指定文件中

第十四章

常见错误和程序调试

1 主要内容

14.1 常见错误分析

14.2 程序调试

§14.1常见错误分析

- (1) 忘记定义变量。
- (2) 输入输出的数据的类型与所用格式说明符不一致。
- (3) 未注意 int 型数据的数值范围。
- (4) 在输入语句 scanf 中忘记使用变量的地址符。
- (5) 输入数据的形式与要求不符。
- (6) 误把“=”作为“等于”运算符。
- (7) 语句后面漏分号。
- (8) 在不该加分号的地方加了分号。
- (9) 对应该有花括号的复合语句，忘记加花括号。
- (10) 括号不配对。

§14.1常见错误分析

- (11) 在用标识符时，忘记了大小写字母的区别。
- (12) 引用数组元素时误用了圆括号。
- (13) 在定义数组时，将定义的“元素个数”误认为是“可使用的最大下标值”。
- (14) 对二维或多维数组的定义和引用的方法不对。
- (15) 误以为数组名代表数组中全部元素。
- (16) 混淆字符数组与字符指针的区别。
- (17) 在引用指针变量之前没有对它赋予确定的值。
- (18) switch语句的各分支中漏写break语句。
- (19) 混淆字符和字符串的表示形式。
- (20) 使用自加（++）和自减（--）运算符时出的错误。

§14.1常见错误分析

- (21) 所调用的函数在调用语句之后才定义，而又在调用前未声明。
- (22) 对函数声明与函数定义不匹配。
- (23) 在需要加头文件时没有用#include命令去包含头文件。
- (24) 误认为形参值的改变会影响实参的值。
- (25) 函数的实参和形参类型不一致。
- (26) 不同类型的指针混用。
- (27) 没有注意函数参数的求值顺序。
- (28) 混淆数组名与指针变量的区别。
- (29) 混淆结构体类型与结构体变量的区别，对一个结构体类型赋值。

§14.1 常见错误分析

- (21) 所调用的函数在调用语句之后才定义，而又在调用前未声明。
- (22) 对函数声明与函数定义不匹配。
- (23) 在需要加头文件时没有用#include命令去包含头文件。
- (24) 误认为形参值的改变会影响实参的值。
- (25) 函数的实参和形参类型不一致。
- (26) 不同类型的指针混用。
- (27) 没有注意函数参数的求值顺序。
- (28) 混淆数组名与指针变量的区别。
- (29) 混淆结构体类型与结构体变量的区别，对一个结构体类型赋值。
- (30) 使用文件时忘记打开，或打开方式与使用情况不匹配。

§14.1 常见错误分析

程序出错有3种情况：

- ① 语法错误
- ② 逻辑错误
- ③ 运行错误

§14.2 程序调试

所谓程序调试是指对程序的查错和排错。调试程序步骤：

- (1) 先进行人工检查，即静态检查。
- (2) 上机调试。
- (3) 在改正语法错误和“警告”后，程序经过连接（link）就得到可执行的目标程序。运行程序，输入程序所需数据，就可得到运行结果。