C语言笔记本

——配谭浩强版《C程序设计》

徐振康记录

2013

记录：徐振康

2013/2/17 Sunday

目录

[说明 7](#_Toc348893285)

[第二章 数据类型、运算符和表达式 8](#_Toc348893286)

[1 基础知识 8](#_Toc348893287)

[1.1 C数据类型 8](#_Toc348893288)

[1.2 注意事项 8](#_Toc348893289)

[1.3符号常量的定义 9](#_Toc348893290)

[2 整型数据 9](#_Toc348893291)

[2.1 整型常量的表示方法 9](#_Toc348893292)

[2.2整型变量的表示方法 10](#_Toc348893293)

[2.2.1整型数据在内存中的存放形式 10](#_Toc348893294)

[2.2.2 整型变量的分类 11](#_Toc348893295)

[2.2.3 整型变量的定义 12](#_Toc348893296)

[2.2.4 整型数据的溢出 13](#_Toc348893297)

[2.3 实型数据 14](#_Toc348893298)

[2.3.1实型常量的表示方法 14](#_Toc348893299)

[2.3.1.1 十进制数形式 14](#_Toc348893300)

[2.3.1.2指数形式 14](#_Toc348893301)

[2.4 实型变量 15](#_Toc348893302)

[2.4.1实型数据在内存中的存放形式 15](#_Toc348893303)

[2.4.2实型变量的分类 16](#_Toc348893304)

[2.4.3实型数据的舍入误差 16](#_Toc348893305)

[2.5字符型数据 17](#_Toc348893306)

[2.5.1字符常量 17](#_Toc348893307)

[2.5.2字符变量 17](#_Toc348893308)

[2.5.3转义字符 18](#_Toc348893309)

[2.5.4字符数据在内存中的存储形式及使用方法 19](#_Toc348893310)

[2.5.5字符串常量 20](#_Toc348893311)

[2.5.6变量赋初值 21](#_Toc348893312)

[2.5.7各类数值型数据之间的混合运算 22](#_Toc348893313)

[2.5.7.1算术运算符和算术表达式 25](#_Toc348893314)

[2.5.7.2赋值运算符和赋值表达式 31](#_Toc348893315)

[2.5.7.2.1赋值运算符 31](#_Toc348893316)

[2.5.7.2.2类型转换 31](#_Toc348893317)

[2.5.7.2.3复合的赋值运算符 33](#_Toc348893318)

[第三章 顺序程序设计——最简单的C程序设计 34](#_Toc348893319)

[1 C基本语句介绍 34](#_Toc348893320)

[2 数据输入输出的概念及在C语言中的实现 36](#_Toc348893321)

[第四章 分支程序设计——最常用的C程序设计 53](#_Toc348893322)

[1 关系运算符和表达式 53](#_Toc348893323)

[2 逻辑运算符及其优先次序 55](#_Toc348893324)

[2.1 if语句 56](#_Toc348893325)

[2.2 条件运算符和条件表达式 68](#_Toc348893326)

[2.3 switch语句 71](#_Toc348893327)

[第四章 循环控制结构程序——最常用的C程序设计2 84](#_Toc348893328)

[1 概述 84](#_Toc348893329)

[2 各种循环 84](#_Toc348893330)

[2.1 goto语句以及用goto语句构成循环 84](#_Toc348893331)

[2.2 while语句 86](#_Toc348893332)

[2.2.1基本语法 86](#_Toc348893333)

[2.2.2使用while语句应注意以下几点 89](#_Toc348893334)

[2.3 do-while语句 90](#_Toc348893335)

[2.4 for语句 92](#_Toc348893336)

[2.4.1 for语句的基本语法 92](#_Toc348893337)

[2.4.2 使用for语句时应注意的问题 93](#_Toc348893338)

[2.5 几种循环的比较 98](#_Toc348893339)

[2.6 break和continue语句 102](#_Toc348893340)

[2.6.1 break语句 102](#_Toc348893341)

[2.6.2 continue语句 105](#_Toc348893342)

[2.6.3 break和continue语句的区别 105](#_Toc348893343)

[第五章 数组 123](#_Toc348893344)

[1 数组的概念 123](#_Toc348893345)

[2 一维数组的定义和引用 124](#_Toc348893346)

[3 一维数组的引用 126](#_Toc348893347)

[4 一维数组的初始化 128](#_Toc348893348)

[4.1 数组初始化的实现方法 129](#_Toc348893349)

[4.2 冒泡排序法(起泡法) 131](#_Toc348893350)

[4.2.1 代码 131](#_Toc348893351)

[4.2.2 过程提示 133](#_Toc348893352)

[5 二维数组的定义和引用 133](#_Toc348893353)

[5.1 二维数组的定义 133](#_Toc348893354)

[5.2 二维数组的引用和初始化 136](#_Toc348893355)

[5.3 其应用(二分法) 149](#_Toc348893356)

[第六章 函数 162](#_Toc348893357)

[1 概述 162](#_Toc348893358)

[2 函数的调用关系 162](#_Toc348893359)

[3 函数参数和函数的值 167](#_Toc348893360)

[3.1 形式参数和实际参数 167](#_Toc348893361)

[3.2 函数的返回值 170](#_Toc348893362)

[4 函数的调用 173](#_Toc348893363)

[4.1 函数调用的一般形式 173](#_Toc348893364)

[4.2 函数的调用方式 175](#_Toc348893365)

[4.2.1 函数语句 175](#_Toc348893366)

[4.2.2 函数表达式 176](#_Toc348893367)

[4.2.3 函数参数 176](#_Toc348893368)

[4.3 对被调用函数的声明和函数原型 176](#_Toc348893369)

[4.3.1 调用另一个函数的条件 176](#_Toc348893370)

[4.3.2 区别关键词 177](#_Toc348893371)

[5 函数的嵌套调用 180](#_Toc348893372)

[5.1 函数调用的释义 180](#_Toc348893373)

[6 递归(属于算法的知识) 185](#_Toc348893374)

[7 数组作为函数参数 192](#_Toc348893375)

[7.1 释义 192](#_Toc348893376)

[7.2 用数组名作函数参数与用数组元素作实参的不同之处 194](#_Toc348893377)

[8 局部变量和全局变量 198](#_Toc348893378)

[8.1 局部变量 198](#_Toc348893379)

[8.2 全局变量 200](#_Toc348893380)

[8.3 全局变量缺点的分析 209](#_Toc348893381)

[9 变量的存储类别 209](#_Toc348893382)

[9.1 动态存储方式与静态存储方式 209](#_Toc348893383)

[9.2 变量的存储类别 210](#_Toc348893384)

[9.2.1 auto变量 211](#_Toc348893385)

[9.2.2 用static声明局部变量 211](#_Toc348893386)

[9.2.2.1 声明方法 211](#_Toc348893387)

[9.2.2.2 对静态局部变量的说明 213](#_Toc348893388)

[9.2.3 register变量 215](#_Toc348893389)

[9.2.4 extern声明外部变量 218](#_Toc348893390)

[9.2.5 用static声明外部变量 221](#_Toc348893391)

[9.3 变量的声明和定义 222](#_Toc348893392)

[9.4 存储类别小结 223](#_Toc348893393)

[10 内部函数和外部函数 225](#_Toc348893394)

[10.1 内部函数 225](#_Toc348893395)

[10.2 外部函数 225](#_Toc348893396)

[第七章 指针 229](#_Toc348893397)

[1 概念和理解 229](#_Toc348893398)

[1.1 概念 229](#_Toc348893399)

[1.2 初识指针 230](#_Toc348893400)

[1.3 指针与指针变量 231](#_Toc348893401)

[1.4 在定义指针变量时要注意两点 232](#_Toc348893402)

[1.5 指针变量的引用“&” 233](#_Toc348893403)

[2 详细说明&和\*运算符 235](#_Toc348893404)

[2.1 释义 235](#_Toc348893405)

[3 数组与指针 244](#_Toc348893406)

[3.1 基本含义 244](#_Toc348893407)

[3.1.1 基本使用方法 244](#_Toc348893408)

[3.1.2 小结 254](#_Toc348893409)

[3.2 多维数组和指针 261](#_Toc348893410)

[3.3 指向多维数组元素的指针变量 265](#_Toc348893411)

[4 字符串与指针 269](#_Toc348893412)

[5 关于a[]和\*a的一些区别 277](#_Toc348893413)

[6 对使用字符指针变量和字符数组的讨论 282](#_Toc348893414)

[7 指向函数的指针 285](#_Toc348893415)

[8 返回指针值的函数 293](#_Toc348893416)

[9 指针函数和函数指针的区别 295](#_Toc348893417)

[10 指针数组的概念 295](#_Toc348893418)

[11 指向指针的指针 300](#_Toc348893419)

[11.1 指向指针数据的指针的定义 300](#_Toc348893420)

[11.2 指针数组作main函数的形参 301](#_Toc348893421)

[11.3 有关指针的数据类型的小结 303](#_Toc348893422)

[11.3.1指针运算小结 303](#_Toc348893423)

[11.3.2 关于void类型 305](#_Toc348893424)

[11.3.2.1 void的作用 305](#_Toc348893425)

[11.3.2.2 void指针和const指针 305](#_Toc348893426)

[11.3.2.3 关于const的三个例子 305](#_Toc348893427)

[第八章 预处理 309](#_Toc348893428)

[1 无参宏定义 309](#_Toc348893429)

[2 带参数的宏定义 316](#_Toc348893430)

[3 文件包含 324](#_Toc348893431)

[4 条件编译 325](#_Toc348893432)

[5 本章小结 330](#_Toc348893433)

[第九章 结构体与共用体 331](#_Toc348893434)

[1 概述 331](#_Toc348893435)

[2 结构体变量的引用 335](#_Toc348893436)

[3 结构体变量的初始化 338](#_Toc348893437)

[4 结构体数组 339](#_Toc348893438)

[5 定义结构体数组 341](#_Toc348893439)

[6 结构体数组的初始化 342](#_Toc348893440)

[7 指向结构类型数据的指针 344](#_Toc348893441)

[8 结构指针变量作函数参数 347](#_Toc348893442)

[9 动态存储分配[重点] 350](#_Toc348893443)

[9.1 malloc函数 350](#_Toc348893444)

[9.2 calloc函数 351](#_Toc348893445)

[9.3 free函数 351](#_Toc348893446)

[9.4 链表 351](#_Toc348893447)

[9.5 建立动态链表 354](#_Toc348893448)

[9.6 实现链表的输出 357](#_Toc348893449)

[9.7 对链表的删除操作 357](#_Toc348893450)

[9.8 对链表的插入操作 365](#_Toc348893451)

[10 共用体 367](#_Toc348893452)

[10.1 共用体的概念 367](#_Toc348893453)

[10.2 共用体和结构体的比较 368](#_Toc348893454)

[10.3 共用体类型数据的特点 369](#_Toc348893455)

[10.4 应用情况 369](#_Toc348893456)

[11 枚举类型 370](#_Toc348893457)

[12 typedef定义类型 374](#_Toc348893458)

[12.1 用typedef声明新的类型名来代替已有的类型名 374](#_Toc348893459)

[12.1.1 声明INTEGER为整型 374](#_Toc348893460)

[12.1.2 声明结构体类型 374](#_Toc348893461)

[12.1.3 声明NUM为整型 375](#_Toc348893462)

[12.1.4 声明STRING为字符指针类型 375](#_Toc348893463)

[12.1.5 声明POINTER为指向函数的指针类型，该函数返回整型值。 375](#_Toc348893464)

[12.2 用typedef定义类型的方法 376](#_Toc348893465)

[第十章 文件操作 378](#_Toc348893466)

[10.1 C文件概述 378](#_Toc348893467)

[10.2 文件的分类 378](#_Toc348893468)

[10.3 ASCII码文件和二进制文件的比较 379](#_Toc348893469)

[10.4 C语言对文件的处理方法 380](#_Toc348893470)

[10.5 文件类型指针 380](#_Toc348893471)

[10.6 文件的打开与关闭 381](#_Toc348893472)

[10.6.1 FILE类型的数组 381](#_Toc348893473)

[10.6.2 文件的打开(fopen函数) 381](#_Toc348893474)

[10.6.3 对于文件使用方式有以下几点说明 382](#_Toc348893475)

[10.6.4 文件的关闭(fclose函数) 384](#_Toc348893476)

[10.7 文件的读写 384](#_Toc348893477)

[10.7.1 概述 384](#_Toc348893478)

[10.7.2 字符读写函数：fgetc和fputc 386](#_Toc348893479)

[10.7.3 字符串的读写函数:fgets和fputs 392](#_Toc348893480)

[10.7.3.1 fgets函数 392](#_Toc348893481)

[10.7.3.2 fputs函数 392](#_Toc348893482)

[10.7.4 数据块读写函数(fread()和fwrinte()) 396](#_Toc348893483)

[10.7.5 格式化读写函数(fprintf()和fscanf()) 401](#_Toc348893484)

[10.7.6 文件输入输出小结 406](#_Toc348893485)

[第十一章 位运算 407](#_Toc348893486)

[1位运算符和位运算 407](#_Toc348893487)

[1.1概念 407](#_Toc348893488)

[2 “按位与”运算符(&) 408](#_Toc348893489)

[2.1 含义 408](#_Toc348893490)

[2.2 按位与运算的用途 408](#_Toc348893491)

[2.2.1 清零 408](#_Toc348893492)

[2.2.2 取一个数的某些指定位 409](#_Toc348893493)

[3 “按位或”运算符(|) 409](#_Toc348893494)

[4 “异或”运算符(^) 411](#_Toc348893495)

[4.1 概念 411](#_Toc348893496)

[4.2 ^运算符应用 412](#_Toc348893497)

[4.2.1 使特定位翻转 412](#_Toc348893498)

[4.2.2 与0相^，保留原值 412](#_Toc348893499)

[4.2.3 交换两个值，不用临时变量 413](#_Toc348893500)

[5 “取反”运算符(~) 413](#_Toc348893501)

[6 左移运算符(<<) 414](#_Toc348893502)

[7 位运算赋值运算符 416](#_Toc348893503)

[8 位运算举例 416](#_Toc348893504)

[附录 421](#_Toc348893505)

# 说明

本笔记配谭浩强版《C程序设计》。您可以在不收取任何费用的前提下分发该笔记，但是严禁用于商业用途。本笔记当中的代码很多出自谭浩强的《C程序设计》，还有大部分是我自己写的，图片大部分出自鱼C工作室，在此说出参考资料的声明，以作免责。

我记录该笔记的目的是：1.为了自己学习C语言，用来记录学习的情况。2.为了分享给大家，以作学习交流。

另外，由于水平很差，所以一定会有很多错误，如果您愿意抽出时间告诉我，我会万分感激。我的联系方式如下：

邮箱：[contorboy@126.com](mailto:contorboy@126.com) [chixuezhihun@163.com](mailto:chixuezhihun@163.com)

祝您学习愉快！

徐振康

2013年2月17日在辽阳

# 第二章 数据类型、运算符和表达式

## 1 基础知识

### 1.1 C数据类型

1. 基本类型
2. 整型
3. 字符型
4. 实型（浮点型）
5. 单精度型
6. 双精度型
7. 枚举类型
8. 指针类型
9. 构造类型
10. 数组类型
11. 结构体类型
12. 共用体类型
13. 空类型

### 1.2 注意事项

习惯上符号常量的标识符用大写字母，变量标识符用小写字母。

### 1.3符号常量的定义

一般形式

#define 标识符 常量

示例：

#define PRICE 30

void main()

{

int num , total;

num = 10;

total = num \* PRICE;

printf(“total=%d\n”,total);

}

## 2 整型数据

### 2.1 整型常量的表示方法

C语言中，二进制，八进制，十进制，十六进制。

1. 十进制整常数：没有前缀，其数码为0~9。（在程序中是根据前缀来区分各种进制数的。）
2. 八进制整常数：开头必须为0，数码为0~7，八进制数通常是无符号数。
3. 十六进制整常数：开头必须是0x或0X，其数码为0~9，A~F或a~f。
4. 整型常数的后缀：在16位字长的机器上，基本整形的长度也为16位，因此表示的数的范围是有限定的，十进制无符号整常数的范围为0~65635，有符号数为-32768~+32767，八进制无符号数的表示范围为0~0177777。十六进制无符号数表示范围0x0~0xffff或0X0~0XFFFF，如果使用的数超过了上述范围，就必须用长整型数来表示。长整型数是用后缀“L”或“l”来表示的。（每位只能存储0,1），16位为可能。

### 2.2整型变量的表示方法

#### 2.2.1整型数据在内存中的存放形式

例如：

int k = 3;

k 变量的名称

3

变量的值

0x0010ff 变量的地址

这里的3在内存中是以二进制的形式存在的。

常识：一个字节1BYTE=8bit

数值是以补码的形式表示的；

○正数的补码和原码相同。

○负数的补码：将该数的绝对值的二进制形式取反再加1。

如：求-10的补码

先求10的原码00001010

取反：11110101

再加1，得-10的补码：11110110（第一位是符号位，1代表负）

#### 2.2.2 整型变量的分类

（注意：这里占多少字节跟系统的和编译器规定有关！可以在编译器上自己试试查查看。可以用如下程序检验

#include<stdio.h>

void main()

{

printf("%d\n",sizeof(int));//若检验short（短整型）可以将“int”改为“short”，还可以检验float，double，long

}

）

|  |  |  |
| --- | --- | --- |
| 类型说明符 | 数的范围 | 字节数 |
| int | -到+ | 4 |
| unsigned int | 0到 | 4 |
| short int | -到+ | 2 |
| unsigned short int | 0到 | 2 |
| long int | -到+ | 4 |
| unsigned long | 0到 | 4 |

每个字节含8位，如int型说明符，字节数为4，则含有的位数为8\*4位，每位有两种数据（0和1）则int型说明符可以表示个数，而int型是有符号的，所以数的范围是从-到+。

1. 基本型：类型说明符为int，在内存中存4个字节。
2. 短整型：类型说明符为short int或short。所占字节和取值范围均与基本型相同。
3. 长整型：类型说明符为long int或long，在内存中占4个字节。
4. 无符号型：类型说明符为unsigned。

#### 2.2.3 整型变量的定义

变量的定义的一般形式为：

类型说明符 变量名标识符，变量名标识符，…；

例如：

int a,b,c;(a,b,c为整型变量)

long x,y;(x,y为长整型变量)

unsigned p,q;(p,q为无符号整型变量)

**在书写变量时，应注意以下几点：**

1. 允许在一个类型说明符后，定义多个相同类型的变量。各变量名之间用“，”号间隔。类型说明符与变量名之间至少用一个空格间隔。
2. 最后一个变量名之后必须以“：”号结尾。
3. 变量定义必须放在变量使用之前。一般放在函数体的开头部分。

示例：整型变量的定义与使用。

#include<stdio.h>

void main()

{

int a,b,c,d;

unsigned u;

a=12;

b=-24;

u=10;

c=a+u;

d=b+u;

printf(“a+u=%d,b+u=%d\n”,c,d);

}

注：若用到printf,scanf等函数，就一定要用到#include<stdio.h>。另外，printf中第一个参数是字符串，后面的参数分别是对应的地址。

#### 2.2.4 整型数据的溢出

示例：整型数据的溢出。

#include<stdio.h>

void main()

{

short a,b;

a=32767;

b=a+1;

printf("%d,%d\n",a,b);

}

32767: 0(符号位+)111111111111111

-32768:1(符号位-)000000000000000

### 2.3 实型数据

#### 2.3.1实型常量的表示方法

实型也称为浮点型。实型常量也称为实数或者浮点数。在C语言中，实数只采用十进制。它有二种形式：十进制小数形式，指数形式。

##### 2.3.1.1 十进制数形式

由数码0~9和小数点组成。

例如：

0.0、25.0、5.789、0.13、5.0、300.、-267.8230

等均为合法的实数。注意，必须有小数点。

##### 2.3.1.2指数形式

由十进制数，加阶码标志“e”或“E”以及阶码（只能为整数，可以带符号）组成。

其一般形式为：

aEn（a为十进制数，n为十进制整数）

如：

2.1E5（等于2.1\*）

3.7E-2（等于3.7\*）

0.5E7（等于0.5\*）

-2.8E-2（等于（-2.8）\*）

以下不是合法的实数：

345（无小数点）

E7（阶码标志E之前无数字）

-5（无阶码标志）

53.-E3（符号位置不对）

2.7E（无阶码）

标准C允许浮点数使用后缀。后缀为“f”或“F”即表示该数为浮点数。如356f和356.是等价的。

### 2.4 实型变量

#### 2.4.1实型数据在内存中的存放形式

实型数据一般占4个字节（32位）内存空间。按指数形式存储。实数3.14159在内存中的存放形式如下：

|  |  |  |
| --- | --- | --- |
| 符号 | 小数部分 | 指数部分 |
| + | .314159 | 1 |

1. 小数部分占的位（bit）数越多，数的有效数字越多，精度越高。
2. 指数部分站的位数越多，则能表示的数值范围越大。

#### 2.4.2实型变量的分类

实型变量分为：单精度（float型）、双精度（double型）和长双精度（long double型）三类。

在Turbo C中单精度型占4个字节（32位）内存空间，其数值范围为3.4E-38~3.4E+38，只能提供七位有效数字。双精度型占8个字节（64位）内存空间，其数值范围为1.7E-308~1.7E+308，可能提供16位有效数字。

|  |  |  |  |
| --- | --- | --- | --- |
| 类型说明符 | 比特数（字节数） | 有效数字 | 数值范围 |
| float | 8\*4 | 前七位 | -到+ |
| double | 8\*8 | 前十六位 | -到+ |
| long double | 8\*8 | 前十六位 | -到+ |

#### 2.4.3实型数据的舍入误差

由于实型变量是由有限的存储单元组成的，因此能提供的有效数字总是有限的。如下例。

例：示例数据的舍入误差。

#include "stdio.h"

void main()

{

float a, b;

a = 123456.789e5;

b = a + 20;

printf("%f\n", a);

printf("%f\n", b);

}

### 2.5字符型数据

字符型数据包括字符常量和字符变量。

#### 2.5.1字符常量

字符常量是用单引号括起来的一个字符。

例如：’a’、’b’、’=’、’+’、’?’都是合法的字符常量。

在C语言中，字符常量有以下特点：

1. 字符常量只能用单引号括起来，不能用双引号或其他括号。
2. 字符常量只能是单个字符，不能是字符串。
3. 字符可以是字符集中任意字符。但数字被定义为字符型之后就不能参与数值运算。如’5’和5是不同的。’5’是字符常量，不能参与运算。

#### 2.5.2字符变量

字符变量是用来存储字符常量，即单个字符。

字符变量的类型说明符是char。字符变量类型定义的格式和书写规则都与整型变量相同。例如：

char a,b;

#### 2.5.3转义字符

转义字符是一种特殊的字符常量。转义字符以反斜线“\”开头，后跟一个或几个字符。转义字符具有特定的含义，不同于字符原有的意义，故称“转义”字符。例如，在前面各例题printf函数的格式串中用到的“\n”就是一个转义字符，其意义是“回车换行”。转义字符主要用来表示那些用一般字符不便于表示的控制代码。

常用的转义字符及其含义

|  |  |  |
| --- | --- | --- |
| 转义字符 | 转义字符的意义 | ASCII代码 |
| \n | 回车换行 | 10 |
| \t | 横向跳到下一制表位置 | 9 |
| \b | 退格 | 8 |
| \r | 回车 | 13 |
| \f | 走纸换页 | 12 |
| \\ | 反斜线符“\” | 92 |
| \’ | 单引号符 | 39 |
| \” | 双引号符 | 34 |
| \v | 竖向制表 |  |
| \a | 鸣铃 | 7 |
| \ddd | 1~3位八进制数所代表的字符 |  |
| \xhh | 1~2位十六进制数所代表的字符 |  |

#### 2.5.4字符数据在内存中的存储形式及使用方法

每个字符变量被分配一个字节的内存空间，因此只能存放一个字符。字符值是以ASCII码的形式存放在变量的内存单元之中的。

如x的十进制ASCII码是120，y的ASCII码是121.对字符变量a,b赋予’x’和’y’值：

a=’x’;

b=’7’;

实际上是在a,b两个单元存放120和55的二进制代码

特殊的ASCII码需要记忆，如A~Z，其对应的ASCII码为65~90。a~z，其对应的ASCII码为97~122，0~9，其对应的ASCII码分别为48~57。注意：上述的ASCII码都是十进制的。

示例：输出字符及对应的ASCII码

#include<stdio.h>

void main()

{

char a, b;

a = 120;

b = 121;

printf("%c, %c\n", a, b);/\*%c用字符型打出。\*/

printf("%d, %d\n", a, b);/\*%d 用整型打出。\*/

}

将小写字母转化成大写字母。

#include<stdio.h>

void main()

{

char a, b;

a = 'a';

b = 'b';

a = a - 32;

b = b - 32;

printf("%c, %c\n%d, %d\n", a, b, a, b);

}

注意：char占1个字节，而int占4个字节，char是int的特殊情况。在这里既可以用char也可以用int。

ASCII码共有256个，其中常用前128个。

#### 2.5.5字符串常量

字符串常量是由一对双引号括起来的字符序列。例如：”CHINA”，”C program”，”$12.5”等都是合法的字符串常量。

字符串常量和字符常量是不同的量。它们之间主要有以下区别：

（补充：char占八位而已！）

1. 字符常量由单引号括起来，字符串常量由双引号括起来。
2. 字符常量只能是单个字符，字符串常量则可以含一个或多个字符。
3. 可以把一个字符常量赋予一个字符变量，但不能把一个字符串常量赋予一个字符变量。

例如：**可以char a = ‘a’不能char a = “a”;**

1. 字符常量占一个字节的内存空间。字符串常量的内存字节数等于字符串中字节数加1。增加的一个字节中存放字符”\0”(ASCII码为0)。这是字符串结束的标志。

#### 2.5.6变量赋初值

在程序中常常需要对变量赋初值，以便使用变量。语言程序中有多种方法为变量提供初值。本小节先介绍在作变量定义的同时给变量赋以初值的方法。这种方法称为初始化。在变量定义中赋初值的一般形式为：

类型说明符 变量1 = 值1, 变量2 = 值2,……;

例如：

int a = 3;

int b, c = 5;

float x = 3.2, y = 3f, z = 0.75;

char ch1 = ‘K’, ch2 = ‘P’;

注意：

1.在变量说明中，不允许连续给多个变量赋初值。如下述说明是错误的：

int a = b = c = 5

必须写为：

int a = 5, b = 5; c = 5;

而赋值语句允许连续赋值。

2.注意赋值表达式和赋值语句的区别。

赋值表达式是一种表达式，它可以出现在任何允许表达式出现的地方，而赋值语句则不能。下述语句是合法的：

if ((x = y + 5) > 0) z = x;

语句的功能是，弱表达式x = y + 5大于0则z = x。

下述语句是非法的：

if ((x = y + 5;) > 0) z = x;

因为x = y + 5;是语句，不能出现在表达式中。

#### 2.5.7各类数值型数据之间的混合运算

变量的数据类型是可以转换的。转换的方法有两种，一种是自动转换，一种是强制转换。自动转换发生在不同数据类型的量混合运算时，由编译系统自动完成。自动转换遵循以下规则：

1. 若参与运算量的类型不同，则先转换成同一类型，然后进行运算。
2. 转换按数据长度增加的方向进行，以保证精度不降低。如int型和long型运算时，先把int量转成long型后在进行运算。
3. 所有的浮点型运算都是以双精度进行的，即使仅含float单精度量运算的表达式，也要先转化成double型，再作运算。
4. char型和short型参与运算时，必须先转换成int型。
5. 在赋值运算中，赋值号两边量的数据类型不同时，赋值号右边量的类型将转换为左边量的类型。如果右边量的数据类型长度左边长时，将丢是一部分数据，这样会降低精度，丢失的部分按四舍五入向前舍入。下图表示了类型自动转换的规则。

char,short(8bit) → int(32bit) → unsigned(64bit) → long(64bit) →double(64bit)

示例：

#include <stdio.h>

void main()

{

float PI = 3.14159;

int s, r = 5;

s = r\*r\*PI;

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

}

本例程序中，PI是实型；s，r为整型。在执行s=r\*r\*PI语时，r和PI都转换成double型计算，结果也为double型。但由于s 为整型，故赋值结果仍为整型，舍去了小数部分。

正确的程序应该如下：

#include <stdio.h>

void main()

{

float PI = 3.14159;

float s;//这里的float可换成double int r = 5;

s = r\*r\*PI;

printf("s = %f\n", s);

}

强制类型转换，是通过类型转换运算来实现的。其一般形式为：

(类型说明符)(表达式)

其功能是把表达式的运算结果强制转换成类型说明符所表示的类型。例如：

(float)a 把a转换成为实型

(int)(x+y) 把x+y的结果转换为整型

在使用强制转换时应该注意以下问题：

1. 类型说明符和表达式都必须加括号（单个变量可以不加括号），如把(int)(x+y)写成(int)x+y则成了把x转换成int型之后再与y相加了。
2. 无论是强制转换或是自动转换，都是为了本次运算的需要而对变量的数据长度进行的临时性转换，而不改变数据说明时对该变量定义的类型。

示例：

#include <stdio.h>

void main()

{

float f = 5.75;

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

}

本例说明，f虽强制转为int型，但只在运算中起作用，是临时的，而f本身的类型并不改变。因此，(int)f的值为5（删去了小数）而f的值仍为5.75。

##### 2.5.7.1算术运算符和算术表达式

1. 加法运算符“+”：加法运算符为双目运算符，即应有两个量参与加法运算。如a+b，4+8等。具有右结合性。
2. 减法运算符“-”：减法运算符为双目运算符。但“-”也可作负值运算符，此时为单目运算，如-x，-5等具有左结合性。
3. 乘法运算符“\*”：双目运算，具有左结合性。
4. 除法运算符“/”：双目运算具有左结合性。参与运算量均为整型时，结果也为整型，舍去小数。如果运算量中有一个是实型，则结果为双精度实型。
5. 求余运算符“%”：双目运算。

例如：

#include <stdio.h>

void main()

{

printf("7 ÷ 2 = %d …… %d\n",7/2,7%2);

}

\*\*注意：“=”赋值运算符是具有左结合性，把后面的值赋给左侧的变量。所以为左结合性。

1. 自增、自减运算符

自增1，自减1运算符：自增1运算符记为“++”，其功能是使变量的值自增1。

自减1运算符记为“--”，其功能是使变量值自减1。

自增1，自减1运算符均为单目运算，都具有右结合性。可有以下几种形式：

++i i自增1后再参与其它运算。

--i i自减1后再参与其它运算。

i++ i参与运算后，i的值再自增1。

i-- i参与运算后，i的值再自减1。

在理解和使用上容易出错的是i++和i--。特别是当它们出现在较为复杂的表达式或语句中时，常常难于弄清，因此应该仔细分析

#include <stdio.h>

void main ()

{

int i = 5, j = 5, p, q;

p = (i++)+(i++)+(i++);/\*此时所有的(i++)的值都是5,但是i的值已经是8了。\*/

q = (++j)+(++j)+(++j);/\*前面的两个(++j)的值都是7,后面的(++j)的值为8。此时的j的值已经是8了。\*/

printf ("%d, %d, %d, %d\n", p, q, i, j);

}

1. 逗号运算符和逗号表达式

在C语言中逗号“，”也是一种运算符，称为逗号运算符。其功能是把两个表达式连接起来组成一个表达式，称为逗号表达式。

期一般形式：

表达式1，表达式2

其求职过程是分别求两个表达式的值，并以表达式2的值作为整个逗号表达式的值。

示例：

#include <stdio.h>

void main ()

{

int a = 2, b = 4, c = 6, x, y;

y = (x = a + b), (b + c);

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

}

对于逗号表达式还要说明两点：

1.逗号表达式一般形式中的表达式1和表达式2也可以又是逗号表达式。

例如：

表达式1，(表达式2，表达式3)

形成了嵌套情形。因此可以把逗号表达式扩展为以下形式：

表达式1，表达式2，…表达式n

整个逗号表达式的值等于表达式n的值。

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

并不是在所有出现逗号的地方都组成逗号表达式，如在变量说明中，函数参数表中逗号只是用作各变量之间的间隔符。

运算符的优先级：

C语言中，运算符的运算优先级共分为15级。1级最高，15级最低。在表达式中，优先级最高的先于优先级低的进行运算。而在一个运算量的两侧的运算符优先级相同时，则按运算符的结合性规定的结合方向处理。

运算符的结合性：

C语言中各运算符的结合性分为两种，即左结合性（自左至右）和右结合性（自右至左）。例如算术运算符的结合性是自左至右，即先左后右。如有表达式x-y+z则y应先与“-”号结合，执行x-y运算，然后再执行+z的运算。这种自左至右的结合方式就称为“左结合性”。而自右至左的结合方向称为“右结合性”。最典型的右结合性运算时赋值运算符。如x=y=z，由于“=”的右结合性，应先执行y=z再执行x=(y=z)运算。C语言运算符中有不少为右结合性，应注意区别，以免理解错误。

※下面是C语言中所使用的运算符的优先级和结合性：

|  |  |  |  |
| --- | --- | --- | --- |
| 优先级 | 运算符 | 运算对象的个数 | 结合性 |
| 最高 | ()(圆括号) [](下标运算符) ->(指向成员运算符) . (结构体、共用体成员运算符) |  | 自左向右 |
|  | !(逻辑非运算符) ~(按位取相反运算符) ++(自增运算符) –(负号运算符) +(正号运算符) (类型)(类型转换运算符) \*(指针运算符) &(取地址运算符) sizeof(求存储长度运算符) | 1  (单目运算符) | 自右向左 |
| \*(乘法运算符) /(除法运算符) %(求余运算符) | 2  (双目运算符) | 自左向右 |
| +(加法运算符) -(减法运算符) | 自左向右 |
| <<（向左移位运算符） >>（向右移位运算符） | 自左向右 |
| < <= > >=  (以上都是关系运算符) | 自左向右 |
| ==(等于运算符) !=(不等于运算符) | 自左向右 |
| &(按位与运算符) | 自左向右 |
| ^(按位异或运算符) | 自左向右 |
| |(按位或运算符) | 自左向右 |
| &&(逻辑与运算符) | 自左向右 |
| ||(逻辑或运算符) | 自左向右 |
| ?:(条件运算符) | 3  (三目运算符) | 自右向左 |
| = += -= \*= /= %= &= ^= |= <<= >>=  (以上都是赋值运算符) | 2  (双目运算符) | 自右向左 |
| 最低 | ,(逗号运算符(顺序求值运算符)) |  | 自左向右 |

##### 2.5.7.2赋值运算符和赋值表达式

###### 2.5.7.2.1赋值运算符

简单赋值运算符和表达式：简单赋值运算符记为“=”。由“=”连接的式子称为赋值表达式。其一般形式为：变量 = 表达式

例如：

x = a + b

w = sin(a) + sin(b)

y = i + j

赋值表达式的功能是计算表达式的值再赋予左边的变量。赋值运算符具有右结合性。因此

a = b = c = 5

可理解为

a = (b = (c = 5))

###### 2.5.7.2.2类型转换

如果赋值运算符两边的数据类型不同，系统将自动进行类型转换，即把赋值号右边的类型转换成左边的类型。具体规定如下：

1. 实型赋予整型，舍去小数部分。前面的例子已经说明了这种情况。
2. 整型赋予实型，数值不变，但将以浮点形式存放，即增加小数部分（小数部分的值为0）。
3. 字符型赋予整型，由于字符型为一个字节，而整型为四个字节（对于visualC++编译器为四个字节，而TurboC编译器为两个字节），故将字符的ASCII码值放到整型量的低八位中，高八位为0。整型赋予字符型，只把低八位赋予字符量。

示例：

#include <stdio.h>

void main ()

{

int a, b = 322;

float x, y = 8.88;

char c1, c2;

a = y;

x = b;

a = c1;

c2 = b;

printf("%d, %f, %d, %c\n", a, x, a, c2);

}

（插一句，%d整型，%f浮点型，%c字符型）

###### 2.5.7.2.3复合的赋值运算符

在赋值符“=”之前加上其它二目运算符可构成复合赋值符。

如+=，-=，\*=，/=，%=,<<=,>>=,&=,^=,!=。

例如：

a+=5 等价于a=a+5

x\*=y+7 等价于x=x\*(y+7)

r%=p 等价于r=r%p

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

# 第三章 顺序程序设计——最简单的C程序设计

## 1 C基本语句介绍

C语句可分为以下五类：

1. 表达式语句：表达式语句由表达式加上分号“;”组成。其一般形式是：表达式;

执行表达式语句就是计算表达式的值。例如：

x = y + z;赋值语句；

y + z;加法运算语句，但是计算结果不能保留，无实际意义；

i++; 自增1语句，i值增1。

1. 函数调用语句：由函数名、实际参数加上分号“;”组成。期一般形式为：函数名(实际参数表);

执行函数语句就是调用函数体并把实际参数赋予函数定义中的形式参数，然后执行被调用函数体中的语句，求取函数值(在后面函数中再详细介绍)。

例如：

printf(“C program”);调用库函数，输出字符串。

1. 控制语句：控制语句用于控制程序的流程，以实现程序的各种结构方式。它们由特定的语句定符组成。C语言有九种控制语句。可以分成以下三类：
2. 条件判断语句：if 语句，switch语句；
3. 循环执行语句：do while语句、while语句、for语句；
4. 转向语句：break语句、goto语句、continue语句、return语句。
5. 复合语句：把多个于今年用括号{}括起来组成的一个语句成复合语句。

在程序中应把复合语句看成是单条语句，而不是多条语句。

例如：

{

x = y + z;

a = b + c;

printf(“%d%d”, x, a);

}

是一条复合语句。

复合语句内的各条语句都必须一份好“;”结尾，在括号“}”外不能加分号。

1. 空语句：只有分号“;”组成的语句称为空语句。空语句是什么也不执行的语句。在程序中空语句可用来空循环体。

例如：

while(getchar()!=’\n’)

{

;

}

本语句的功能是，只要从键盘输入的字符不是回车则重新输入。

这里的循环体为空语句。

## 2 数据输入输出的概念及在C语言中的实现

1)所谓输入输出是以计算机为主体而言的。

2)本章介绍的是向标准输出设备显示器输出数据的语句。

3)在C语言中，所有的数据输入/输出都是由库函数完成的。因此都是函数语句。

4)在使用C语言库函数时，要用预编译命令#cinclude 将有关“头文件”包括到源文件中。

使用标准输入输出库函数时要用到“stdio.h”文件，因此源文件开头应有以下预编译命令：

#include <stdio.h>或#include “stdio.h”（搜索用户自己写的头文件时用“”，搜索系统中本来有的头文件时用<>,这样会提高运行效率）

stdio是standard input & output的意思。

5)考虑到printf和scanf函数使用频繁，系统允许在使用这两个函数时刻不加#include <stdio.h>或#include “stdio.h”

6)字符数据的输入输出

putchar 函数(字符输出函数)

putchar 函数时字符输出函数，其功能是在显示器上输出单个字符。

其一般形式是：

putchar (字符变量)

例如：

putchar (‘A’);(输出大写字母A)

putchar (x);(输出字符变量x的值)

putchar (‘\101’);(也是输出字符A)

putchar (‘\n’);(换行)

对控制字符则执行控制功能，不在屏幕上显示。

使用本函数前必须要用文件包含命令：

#include <stdio.h>或#include “stdio.h”

示例：

#include<stdio.h>

void main ()

{

char a = 'B', b = 'o', c = 'k';

putchar(a);

putchar(b);

putchar(b);

putchar(c);

putchar('\t');

putchar(a);

putchar(b);

putchar('\n');

putchar(b);

putchar(c);

}

getchar函数(键盘输入函数)

getchar函数的功能是从键盘上输入一个字符。

期一般形式是：

getchar();

通常把输入的字符赋予一个字符变量，构成赋值语句，如：

char c;

c = getchar();

示例：

输入单个字符。

#include<stdio.h>

void main ()

{

char c;

printf("input a character\n");

c = getchar();

putchar(c);

}

7)格式输入与输出

printf 函数(格式输出函数)

printf 函数称为格式输出函数，其关键字最末一个字母f即为“格式”(format)之意。其功能是按用户指定的格式，把指定的数据显示到显示器屏幕上。在前面的例题中我们已多次使用过这个函数。

1.printf函数调用的一般形式

printf函数是一个标准库函数，它的函数原型在头文件“stdio.h”中。但作为一个特例，不要求在使用printf函数之前必须包含stdio.h文件。

printf函数调用的一般是形式为：

printf(“格式控制字符串”,输出表列)

其中格式控制字符串用于指定输出格式。格式控制串可由格式字符串和非格式字符串两种组成。格式字符串是以%开头的字符串，在%后面跟有各种格式字符串，以说明输出数据的类型、形式、长度、小数位数等。如：

“%d”表示按十进制整型输出；

“%ld”表示十进制长整型输出；

“%c”表示字符型输出等。

非格式字符串在输出时原样照印，在显示中起提示作用。

输出表列中给出了各个输出项，要求格式字符串和各输出项在数量和类型上应该一一对应。

示例：

#include<stdio.h>

void main ()

{

int a = 88,b = 89;

printf ("%d %d\n",a,b);

printf ("%d %d\n",a,b);

printf ("%c %c\n",a,b);

printf ("a = %d, b = %d\n",a,b);

}

本例中四次输出了a，b的值，但由于格式控制串不同，输出的结果也不同。第四行的输出语句格式控制串中，量格式串%d之间加了一个空格(非格式字符)，所以输出的a，b值之间有一个空格。第五行的printf语句格式控制串中加入的是非格式字符逗号，因此输出的a，b值之间加了一个逗号。第六行的格式串要求按字符型输出a，b值。低七行中为了提示输出结果又增加了非格式字符串。

|  |  |
| --- | --- |
| 格式字符 | 意义 |
| d | 以十进制形式输出带符号整数(正数不输出符号) |
| o | 以八进制形式输出无符号整数(不输出前缀0) |
| x,X | 以十六进制形式输出无符号整数(不输出前缀0x) |
| u | 以十进制形式输出无符号整数 |
| f | 以小数形式输出单、双精度实数 |
| e,E | 以指数形式输出单、双精度实数 |
| g,G | 以%f或%e中较短的输出宽度输出单、双精度实数 |
| c | 输出单个字符 |
| s | 输出字符串 |

2.标志：标志字符为-、+、#、空格四种，其意义下表所示：

|  |  |
| --- | --- |
| 标志 | 意义 |
| - | 结果左对齐，右边填空格 |
| + | 输出符号(正号或负号) |
| 空格 | 输出值为正时冠以空格，为负时冠以负号 |
| # | 对c,s,d,u类无影响；对o类，在输出时加前缀o；对x类，在输出时加前缀ox；对e,g,f类当结果有小数时才给出小数点 |

3.输出最小宽度：用十进制整数来表示输出的最少位数。若实际位数多于定义的宽度，则按实际位数输出，若实际位数少于定义的宽度则补以空格或O。

4.精度：精度格式符以“.”开头，后跟十进制整数。本项的意义是：如果输出数字，则表示小数的位数；如果输出的是字符，则表示输出字符的个数；若实际位数大于所定义的精度数，则截去超过的部分。

5.长度：长度格式符为h，l两种，h表示按短整型量输出，l表示按长整型量输出。

示例：

#include<stdio.h>

void main ()

{

int a = 15;

float b = 123.1234567;

double c = 12345678.1234567;

char d = 'p';

printf("a = %d, %5d, %o,%x\n",a,a,a,a);

printf("b = %f, %lf, %5.4lf, %e\n",b,b,b,b);/\*5.4表示输出字符的整数部分为5位，小数部分为4位。\*/

printf("c = %lf, %f, %8.4lf\n",c,c,c);

printf("d = %c, %8c\n",d,d);

}

使用printf函数时还要注意一个问题，那就是输出表列中的求值顺序。不同的编译系统不一定相同，可以从左到右，也可从右到左。Turbo C和visual C++是按从右到左进行的。请看下面的两个例子：

示例：

#include<stdio.h>

void main ()

{

int i = 8;

printf("%d\n%d\n%d\n%d\n%d\n%d\n",++i,

--i,i++,i--,-i++,-i--);

}

#include<stdio.h>

void main ()

{

int i = 8;

printf("%d\n",++i);

printf("%d\n",--i);

printf("%d\n",i++);

printf("%d\n",i--);

printf("%d\n",-i++);

printf("%d\n",-i--);

}

这两个程序的区别是用一个printf语句和多个printf语句输出。但从结果可以看出是不同的！！

scanf函数(格式输入函数)

scanf函数称为格式输入函数，即按用户指定的格式从键盘上把数据输入到指定的变量之中。

1.scanf函数的一般形式

scanf函数是一个标准库函数，它的函数原型在头文件“stdio.h”中，与printf函数相同，C语言也允许在使用scanf函数之前不必包含stdio.h文件。

scanf(“格式控制字符串”,地址表列);

其中，格式控制字符串的作用与printf函数相同，但不能显示非格式字符串，也就是不能显示提示字符串。地址表列中给出各变量的地址。地址是由地址运算符“&”后跟变量名组成的。

例如：

&a,&b

分别表示变量a和变量b的地址。

这个地址就是编译系统在内存中给a,b变量分配的地址。在C语言中，使用了地址这个概念，这是与其它语言不同的。应该是把变量的值和变量的地址这两个不同的概念区别开来。变量的地址是C编译系统分配的，用户不必关心具体的地址是多少。

变量的地址和变量值的关系如下：

在赋值表达式中给变量赋值，如：

a = 567

则，a为变量名，567是变量的值，&a是变量的地址。

但在赋值号左边是变量名，不能写地址，而scanf函数在本质上也是给变量赋值，但要求变量的地址，如&a。这两者在形式上是不同的。&是一个取地址运算符，&a是一个表达式，其功能是求变量的地址。

示例：

#include <stdio.h>

void main ()

{

int a, b, c;

printf("input a, b, c\n");

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

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

}

2.格式字符串

格式字符串的一般形式为：

%[\*][输入数据的宽度][长度]类型

其中右括号[]的项。各项的意义如下：

1)类型：表示输入数据的类型，其格式符合意义如下表所示。

|  |  |
| --- | --- |
| 格式 | 字符意义 |
| d | 输入十进制整数 |
| o | 输入八进制整数 |
| x | 输入十六进制整数 |
| u | 输入无符号十进制整数 |
| f或e | 输入实型数(用小数形式或指数形式) |
| c | 输入单个字符 |
| s | 输入字符串 |

2)“\*”符：用以表示该输入项，读入后不赋予相应的变量，即跳过该输入值。

如：scanf(“%d%\*d%d”,&a,&b);

当输入为：1 2 3时，把1赋予a， 2被跳过， 3赋予b。

3)宽度：用十进制整数指定输入的宽度(即字符数)。

如：scanf(“%5d”,&a);

输入：12345678

只把12345赋予变量a，其余部分被截去。

如：

scanf(“%4d%4d”,&a,&b);

输入：12345678

将把1234赋予a，而把5678赋予b。

4)长度：长度格式符为l和h，l表示输入长整型数据(如%ld)和双精度浮点数(如%lf)。H表示输入短整型数据。

使用scanf函数还须注意以下几点：

1. scanf函数中没有精度控制，如：scanf(“%5.2f”,&a);是合法的。不能企图用到此语句输入小数位2位的实数。
2. scanf中要求给出变量的地址，如给出变量名则会出错。如scanf(“%d”,a);是非法的，应该为scanf(“%d”,&a);才是合法的。
3. 在输入多个数据时，若格式控制串中没有非格式字符作输入数据之间的间隔则可用空格，Tab或回车作间隔。C编译在碰到空格，Tab，回车或非法数据(如对”%d”输入”12A”时，A即为非法数据)时即认为该数据结束。
4. 在输入字符数据时，若格式控制串中无非格式字符，则认为所有输入的字符均为有效字符。

例如：scanf(“%c%c%c”,&a,&b,&c);

输入为：

d e f

则把’d’赋予a,’’赋予b,’e’赋予c。

只有当输入为：def时，才能把’d’赋予b,’f’赋予c。

如果在格式控制中加入空格作为间隔，

如：scanf(“%c %c %c”,&a,&b,&c);

则输入时各数之间可加空格。

示例：

#include <stdio.h>

void main ()

{

char a,b;

printf("input charater a and b:\n");

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

printf("%c%c\n", a, b);

}

1. 如果格式控制串中有非格式字符则输入时也要输入该非格式字符。

例如：

scanf(“%d,%d,%d”,&a,&b,&c);

其中用非格式符“，”作间隔符，故输入时应为：

5,6,7

又如：

scanf(“a=%d,b=%d,c=%d”,&a,&b,&c);

则输入应为：

a=5,b=6,c=7

1. 如输入的数据与输出的类型不一致时，虽然编译能通过，但结果将不正确。

例题1：输入三个小写字母，输入其ASCII码和其对应的大写字母。

#include <stdio.h>

void main()

{

char a, b, c;

printf("输入三个小写字母\n");

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

printf("%d,%d,%d\n%c,%c,%c\n",a,b,c,a-32,b-32,c-32);

}

例题2：输出各种数据类型的字节长度。

#include <stdio.h>

void main()

{

int a;

long b;

float f;

double d;

char c;

printf("\nint:%d\nlong:%d\nfloat:%d\ndouble:%d\ nchar:%d\n",sizeof(a),

sizeof(b),sizeof(f),sizeof(d),sizeof(c));

}

例题3：输入三角形的三边长，求三角形面积。

已知三角形的三边长a,b,c。则三角形的面积公式为：

area =

其中s = (a + b + c)/2

#include <stdio.h>

#include <math.h>

void main()

{

double a, b, c, s, area;

printf("请输入三边的边长，按a，b，c的顺序，用逗号隔开,按回车键结束输入。\n");

scanf("%lf,%lf,%lf",&a,&b,&c);

if ((a+b<=c)||(b+c<=a))

{

printf("您输入的边长不能构成一个三角形。\n");

return;

}

if (a+c<=b)

{

printf("您输入的边长不能构成一个三角形。\n");

return;

}

area = sqrt(s\*(s-a)\*(s-b)\*(s-c));

s = (a + b + c)/2.0;

printf("该三角形的三边分别为：%7.2f, %7.2f, %7.2f \n\n面积为：%7.2f \n\n", a, b, c,s);

}

例题3：求a+bx+c=0方程的根，a,b,c由键盘输入，设>0。求根公式为：

p =

令

则

#include <stdio.h>

#include <math.h>

void main()

{

double x1, x2, a, b, c;

printf("请输入各项的系数，降幂排列a,b,c,用逗号隔开,按回车键结束输入。\n");

scanf("%lf,%lf,%lf",&a,&b,&c);

if(b\*b-4\*a\*c<0)

{

printf("您输入的各项系数所构成的方程无实数根。\n");

return;

}

x1 = (-b+sqrt(b\*b-4.0\*a\*c))/(2.0\*a);

x2 = (-b-sqrt(b\*b-4.0\*a\*c))/(2.0\*a);

printf("您输入的二次项系为：%f，一次项系数为：%f，常数项为：%f。\n\n根分别为:x1 = %f\n x2 = %f\n\n", a, b, c, x1, x2);

}

# 第四章 分支程序设计——最常用的C程序设计

## 1 关系运算符和表达式

关系运算符：在程序中经常需要比较两个量的大小关系，以决定程序下一步的工作。比较两个量的运算符称为关系运算符。

在C语言中有以下关系运算符：

1. < 小于
2. <= 小于或等于
3. > 大于
4. >= 大于或等于
5. == 等于
6. != 不等于

关系运算符都是双目运算符，其结合性均为左结合。关系运算符的优先级低于算术运算符，高于赋值运算符。在六个关系运算符中，前四个<,<=,>,>=的优先级相同，高于==和!=,==和!=的优先级相同。

关系表达式：

关系表达式的一般形式为：

表达式 关系运算符 表达式

例如：

a + b > c – d

x > 3/2

‘a’+ 1 < c

-i – 5\*j == k + 1

都是合法的关系表达式。由于表达式也可以又是关系表达式。因此也允许出现嵌套的情况。例如：

a > ( b > c )

a != ( c == d )等。

关系表达式的值是“真”和“假”，用“1”和“0”表示。

如：

5>0的值为“真”，即为1。（若i=(5>0),则i=1。）

(a=3)>(b=5)由于3>5不成立，故其值为“假”，即为0。

示例：

#include <stdio.h>

void main()

{

char c = 'k';

int i = 1, j = 2, k = 3;

float x = 3e+5, y = 0.85;/\*注意：这里的3e+5的字符之间不可以加空格。否则报错。因为3e+5是表示3乘以10的五次方。\*/

printf("%d, %d\n", 'a' + 5 < c, -i - 2 \* j >= k + 1);

printf("%d, %d\n", 1 < j < 5, x - 5.25 <= x + y);

printf("%d, %d\n", i + j + k == -2 \* j, k == j == i + 5);

}

在本例中求出了各种关系运算符的值。字符变量是以它对应的ASCII码参与运算的。对于含多个运算符的表达式，如k==j==i+5,根据运算符的左结合性，先计算k==j,该式不成立，其值为0，再计算0==i+5，也不成立，故表达式值为0。

## 2 逻辑运算符及其优先次序

C语言中提供了三种逻辑运算符：

1. && 与运算
2. || 或运算
3. ! 非运算

与运算符&&和或运算||均为双目运算符。具有左结合性。非运算符!为单目运算符，具有右结合性。逻辑运算符和其它运算符优先级的关系可表达如下：

!(非)→&&(与)→||(或)（注意：3&&5的返回值为1，因为3是非零的，5也是非零的。经过与运算后返回值为1。同理其它的逻辑运算符。而3&&0的返回值为0，因为3是非零的，0表示假的，经过与运算后返回值为0。若i的值是5，则!i为0。）

逻辑表达式

逻辑表达式的一般形式为：

表达式 逻辑运算符 表达式

其中的表达式可以又是逻辑表达式，从而组成了嵌套的情形。

例如：

(a&&b)&&c

根据逻辑运算符的左结合性，上式也可写为：

a&&b&&c

逻辑表达式的值是式中各种逻辑运算的最后值，以“0”和“1”分别代表“真”和“假”。

### 2.1 if语句

用if语句可以构成分支结构。它根据给定的条件进行判断，以决定执行某个分支程序段。C语言的if语句有三种基本形式。

if语句三种形式

第一种形式为基本形式：

if(表达式) 语句

其语义是：如果表达式的值为真，则执行其后的语句，否则不执行该语句。其过程可表示为下图。

表达式

语句

真(非0) 假(0)

示例：

#include <stdio.h>

void main()

{

int a, b, max;

printf("\n input two numbers:");

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

max = a;

if (max < b)

{

max = b;

}

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

}

第二种形式为：if-else

if(表达式)

语句1；

else

语句2；

表达式

真 假

语句2

语句1

示例：

#include <stdio.h>

void main()

{

int a, b;

printf("input two numbers: ");

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

if(a>b)

{

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

}

else

{

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

}

}

第三种形式为：if-else-if形式

前二种形式的if语句一般都用于两个分支的情况。当有多个分支选择时，可采用if-else-if语句，其一般形式为：

if(表达式1)

语句1;

else if(表达式2)

语句2;

else if(表达式3)

语句3;

…

else if(表达式m)

语句m;

else

语句n;

示例：

#include <stdio.h>

void main()

{

char c;

printf("input a character:");

c = getchar();

if (c<32)

{

printf("This is a control character\n");

}

else if(c>='0'&&c<='9')

{

printf("This is a figit\n");

}

else if(c>='A'&&c<='Z')

{

printf("This is a capital letter\n");

}

else if(c>='a'&&c<='z')

{

printf("This is a small letter\n");

}

else

{

printf("This is an other character\n");

}

}

使用if语句时应该注意以下问题：

1. 在三种形式的if语句中，在if关键字之后均为表达式。该表达式通常是逻辑表达式或关系表达式，但也可以是其它表达式，如赋值表达式等，甚至也可以是一个变量。

例如：

if(a=5)语句；

if(b)语句；

都是允许的。只要表达式的值为非0，即为“真”。

示例：

#include <stdio.h>

void main()

{

int a;

a = 3;

if( a == 3 )/\*此处的a == 3很容易出错。如果在写程序时忘记打一个等于号，那么编译器不能查出错误，一律当作正确处理，因为少了一个等于号就是代表把3赋值给a，那么判断条件都为真。因此，此处可以写成3 == a，这样若少打一个等于号，编译器便会查出错误。\*/

{

printf("a equals 3!!\n");

}

else

{

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

}

}

1. 在if语句中，条件判断表达式必须用括号括起来，在语句结束之后必须加分号。
2. 在if语句的三种形式中，所有的语句应为单个语句，如果要想在满足条件时执行一组(多个)语句，则必须把这一组语句用{}括起来组成一个复合语句。但要注意的是在}之后不能再加分号。

例如：if(a > b)

{

a++;

b++;

}

else

{

a = 0;

b = 10;

}

补充例题1：写一个程序完成下列功能：

1. 输入一个分数score
2. score<60 输入E
3. 60<=score<70 输入D
4. 75<=score<80 输入C
5. 80<=score<90 输入B
6. 90<=score 输入A

#include <stdio.h>

void main()

{

int score;

printf("请输入分数：");

scanf("%d",&score);

if(100 < score)

{

printf("输入错误。");

return;

}

if(score < 60)

{

printf("E\n");

}

else if((60 < score || 60 == score)&& 70 > score)

{

printf("D\n");

}

else if((70 < score || 70 == score)&& 80 > score)

{

printf("C\n");

}

else if((80 < score || 80 == score)&& 90 > score)

{

printf("B\n");

}

else

{

printf("A\n");

}

}

补充例题2：输入三个数a,b,c,要求按由小到大的顺序输出。

#include <stdio.h>

void main()

{

int a, b, c, d;

printf("Please input three numbers:");

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

if (a > b)

{

a = d;

d = b;

b = a;

}

if (a > c)

{

a = d;

d = c;

c = a;

}

if (b > c)

{

b = d;

d = c;

c = b;

}

printf("%d,%d,%d\n",a,b,c);

}

if语句的嵌套：

当if语句中的执行语句又是if语句时，则构成了if语句嵌套的情形。

其一般形式为：

if()

{

if()

……;

}

if(表达式)

if语句；

或者为

if()

if() 语句1

else 语句2

else

if() 语句3

else 语句4

if(表达式)

if语句；

else

if语句；

编程工作者把嵌套把握在四个到五个，这样会保证逻辑有条理。

在嵌套内的if语句可能又是if-else型的，这将会出现多个if和多个else重叠的情况，这时要特别注意if和else的配对问题。

例如：

if(表达式1)

if(表达式2)

语句1；

else

语句2；

其中的else究竟是与哪一个if配对呢？

应该理解为：

if(表达式1)

if(表达式2)

语句1；

else

语句2；

还是应理解为：

if(表达式1)

if(表达式2)

语句1；

else

语句2；

为了避免这种二义性，C语言规定，else总是与它前面最近的if配对，因此对上述例子应按前一种情况理解。即就近原则。

采用嵌套结构实质上是为了进行多分支选择，实际上有多种选择。这种问题用if-else-if语句也可以完成。而且程序更加清晰。因此，在一般情况下较少适用if语句嵌套结构。以使程序更便于阅读理解。

### 2.2 条件运算符和条件表达式

条件运算符为：“？”和“：”，它是C语言中的唯一一个三目运算符，即有三个参与运算的量。

由条件运算符组成条件表达式的一般形式为：

表达式1？ 表达式2：表达式3

其求值规则为：如果表达式1的值为真，则以表达式2的值作为条件表达式的值，否则以表达式2的值作为整个条件表达式的值。

条件表达式通常用于赋值语句之中。

例如条件语句：

if(a>b) max = a;

else max = b;

可引用条件表达式写为：

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

执行该语句的语义是：如果a>b为真，则把a赋予max,否则把b赋予max。

使用条件表达式时，还应注意以下几点：

1. 条件运算符的运算优先级低于关系运算符和算术运算符，但高于赋值符。因此max=(a>b)?a:b可以去掉括号而写为max=a>b?a:b
2. 条件运算符？和：是一对运算符，不能分开单独使用。
3. 条件运算符的结合方向是自右至左。例如：
4. b?a:c>d?c:d

应理解为a>b?a:(c>d?c:d)

这也就是条件表达式嵌套的情形，即其中的表达式3又是一个条件表达式。

示例：

#include <stdio.h>

void main()

{

int a, b, max;

printf("\ninput two numbers:\n");

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

printf("max = %d", a>b?a:b);

}

补充：输入一个字符，判断它是否大写字母，如果是，将它转换成小写字母；如果不是，不转换。然后输出最后得到的字符。

#include <stdio.h>

void main()

{

char a;

int f;

printf("input a character:\n");

scanf("%c",&a);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

printf("%d\n",a);//这里的a是用户输入的字符的ASCII码，是一个数字。

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

f = (a>=65&&a<=90)?a+32:a;

printf("%c\n",f);

}

### 2.3 switch语句

C语言还提供了另一种用于分支选择的switch语句，期一般形式为：

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；

}

其语义是：计算表达式的值。并逐个与其后的常量表达式值相比较，当表达式的值与某个常量表达是的值相等时，即执行其后的语句，然后不再进行判断，继续执行后面所有case后的语句。如表达式的值与所有case后的常量表达式均不相同时，则执行default后的语句。

示例：

#include <stdio.h>

void main()

{

int a;

printf("input integer number:");

scanf("%d",&a);

switch (a)

{

case 1:printf("Monday\n");break;/\*这里的break；是专门用于跳出该switch语句的。\*/

case 2:printf("Tuesday\n");break;

case 3:printf("Wendnesday\n");break;

case 4:printf("Thursday\n");break;

case 5:printf("Friday\n");break;

case 6:printf("Saturday\n");break;

case 7:printf("Sunday\n");break;

default:printf("errer\n");

}

}

在使用switch语句时还应注意一下几点：

1. 在case后的各常量表达式的值不能相同，否则会出现错误。
2. 在case后，允许有多个语句，可以不用{}括起来。（但最好用{}括起来，看起来会比较有条理。）
3. 各case和default子句的先后顺序可以变动，而不会影响程序执行结果。
4. default子句可以忽略不用。

例题1：输入三个整数，输出最大数和最小数。

可以用如下的程序来实现。

#include <stdio.h>

void main()

{

int a, b, c, temp1, temp2;

printf("Please input three numbers:\n");

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

temp1 = a>b?a:b;

temp2 = temp1>c?temp1:c;

printf("The max is %d.\n",temp2);

temp1 = a>b?b:a;

temp2 = temp1>c?c:temp1;

printf("The min is %d.\n",temp2);

}

也可以用if语句：

#include <stdio.h>

void main()

{

int a, b, c, max, min;

printf("input three numbers:");

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

if(a>b)

{

max = a;

min = b;

}

else

{

max = b;

min = a;

}

if(max<c)

{

max = c;

}

else if(min>c)

{

min = c;

}

printf("max = %d\nmin = %d\n",max,min);

}

(**小技巧：如果写完的程序代码很乱，没有条理，可以全部选中，然后按Alt+F8，可以调整代码的布局使其整齐。**)

例题2：

计算器程序。用户输入运算数和四则运算符，输出计算结果。

#include <stdio.h>

void main()

{

double a, b;

char sign;

printf("按如下格式进行输入：a+(-,\*,/)b=\n");

scanf("%lf %c %lf",&a,&sign,&b);

switch(sign)

{

case '+': printf("%4.2f %c %4.2f = %4.2f\n", a, sign, b, a + b); break;

case '-': printf("%4.2f %c %4.2f = %4.2f\n", a, sign, b, a - b); break;

case '\*': printf("%4.2f %c %4.2f = %4.2f\n", a, sign, b, a \* b); break;

case '/': if (0 == b)

{

printf("对不起，0不能做分母。\n");

return;

}

printf("%4.2f %c %4.2f = %4.2f\n", a, sign, b, a / b); break;

default: printf("输入错误。\n");

}

}

例题3：

输入一个年份，并判断是否为闰年？

#include <stdio.h>

void main()

{

int year;

printf("请输入一个年份:\n");

scanf("%d",&year);

if (0>year)

{

printf("输入错误\n");

return;

}

if (year %100 == 0)

{

if(year % 400 == 0)

{

printf("该年份为闰年。\n");

}

else

{

printf("该年份为平年。\n");

}

}

else if (year % 4 == 0)

{

printf("该年份为闰年。\n");

}

else

{

printf("该年份为平年。\n");

}

}

例题4：

运输公司对用户计算运费。

路程(s)越远，每公里运费越低。标准如下（单位：km）：

s<250 没有折扣

250<=s<500 2%折扣

500<=s<1000 5%折扣

1000<=s<2000 8%折扣

2000<=s<3000 10%折扣

3000<=s 15%折扣

设每公里每吨货物的基本运费为p，货物重为w，距离为s，折扣为d，则总运费f的计算公式为：

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

可以采用下面的程序：

#include <stdio.h>

void main()

{

double p, w, s, d;

printf("请输入公里数、货物重、基本运费：\n");

scanf("%lf%lf%lf",&s,&w,&p);

if(s<0)

{

printf("输入错误。\n");

return;

}

if(s<250)

{

d = 0;

}

if (s>=250&&s<500)

{

d = 0.02;

}

if (s>=500&&s<1000)

{

d = 0.05;

}

if (s>=1000&&s<2000)

{

d = 0.08;

}

if (s>=2000&&s<3000)

{

d = 0.10;

}

else

{

d = 0.15;

}

printf("总运费为：%10.2f\n",p\*w\*s\*(1-d));

}

也可以采用下面的程序：

#include <stdio.h>

void main()

{

int c, s;

double p, w, d, f;

printf("请输入公里数、货物重、基本运费：\n");

scanf("%lf%lf%lf",&s,&w,&p);

if( s>=3000 )

{

c = 12;

}

else

{

c = s/250;

}

switch(c)

{

case 0:

d = 0;

break;

case 1:

d = 2;

break;

case 2:

case 3:

d = 5;/\*此句表示若,满足case2或case3，都执行d = 5;这条语句。\*/

break;

case 4:

case 5:

case 6:

case 7:

d = 8;

break;

case 8:

case 9:

case 10:

case 11:

d = 10;

break;

case 12:

d = 15;

break;

}

f = p\*w\*s\*(1-d/100.0);

printf("总运费为：%15.4f\n",f);

}

例题5：

一个整数，它加上100后是一个完全平方数，在加上168又是一个完全平方数，请问该数是多少?

程序分析：在10万以内判断，先将该数加上100后再开方，再将该数加上268后再开方，如果开放后的结果满足如下条件，即是结果。

例题6：

输入某年某月某日，判断这一天是这一年的第几天？

程序分析：以3月5日为例，应该先把前两个月的加起来，然后再加上5天即本年的第几天，特殊情况，闰年且输入月份大于3时需考虑多加一天。

.

# 第四章 循环控制结构程序——最常用的C程序设计2

概述

什么是循环？

为什么要是用循环？

## 1 概述

循环结构是程序中一种很重要的结构。其特点是，在给定条件成立时，反复执行某程序段，直到条件不成立为止。给定的条件称为循环条件，反复执行的程序段称为循环体。C语言提供了多种循环语句，可以组成各种不同形式的**循环结构**。

1. 用goto语句和if语句构成循环；
2. 用while语句;
3. 用do-while语句；
4. 用for语句；

## 2 各种循环

### 2.1 goto语句以及用goto语句构成循环

goto语句是一种无条件转移意语句，与BASIC中的goto语句相似。goto语句的使用格式为：

goto 语句标号；

其中标号是一个有效的标识符，这个标识符加上一个“：”一起出现在函数内某处，执行goto语句后，程序间跳转到该标号处并执行其后的语句。另外标号必须与goto语句同处于一个函数中，但可以不在一个循环层中。通常goto语句与if条件语句连用，当满足某一条件时，程序跳到标号处运行。

但是注意：goto语句通常不用，主要因为它将使程序层次不清，且不易读，但在多层嵌套退出时，用goto语句则比较合理。

示例：用goto语句和if语句构成循环

#include <stdio.h>

void main()

{

int i, sum = 0;

i = 1;

loop:if(i<=100)

{

sum = sum + i;

i++;

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

goto loop;

}

printf("\nThe final result is %d.\n",sum);

}

### 2.2 while语句

#### 2.2.1基本语法

while语句的一般形式为：

while(表达式)语句

其中表达式是循环条件，语句为循环体。

while语句的语义是：计算表达式的值，当值为真(非零)时，执行循环体语句。其执行过程可由下图表示。

语句

表达式

0

非0

**注意：如果表达式的值一开始就为0，则语句一次也会被不执行。**

示例：用while语句求

#include <stdio.h>

void main()

{

int i, sum = 0;

i = 1;

while(i<=100)

{

sum = sum + i;

i++;

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

}

printf("\nThe final result is %d.\n",sum);

}

注意：这样子循环永远也不会退出！

#include <stdio.h>

void main()

{

int i, sum = 0;

i = 1;

while(i<=100)

//{

sum = sum + i;

i++;

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

//}

printf("\nThe final result is %d.\n",sum);

}

这就是传说中的死循环！！！

若运行改程序，单核处理器会占用将近100%的CPU！！！双核处理器会占用50%的CPU！！！所以，不要将程序弄成死循环！！！！

示例：统计从键盘上输入的一行字符的个数。

#include <stdio.h>

void main()

{

int i = 0;

printf("Please input a string:\n");

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

{

i++;

}

printf("\nThe string contains %d character(s).\n",i);

}

#### 2.2.2使用while语句应注意以下几点

1. while语句中的表达式一般是关系表达式或逻辑表达式，只要表达式的值为真(非0)即可继续循环。

示例：

#include <stdio.h>

void main()

{

int a = 0, n;

printf("\ninput n:");

scanf("%d",&n);

while(n--)//当()中的值为0时不再执行while语句中的循环体。

{

printf("%d\n",a++\*2);//a++\*2相当于a\*2;a++;

}

}

1. 循环体如包括有一个以上的语句，则必须用{}括起来，责成复合语句。

### 2.3 do-while语句

do-while语句的一般形式为：

do

语句

while(表达式);

这个循环与while循环的不同在于：它先执行循环中的语句，然后再判断表达式是否为真，如果为真则继续循环；如果为假，则终止循环。因此，do-while循环至少要执行一次循环语句。

while流程图

i≤100

sum=sum+1;

i=i+1

i=1

非0 0

do-while流程图

表达式

循环体语句

非0

0

示例：

#include <stdio.h>

void main()

{

int i = 1, sum = 0;

do

{

sum = sum + i;

i++;

}while(i<=100);

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

}

while和do-while循环比较

do-while语句要先执行do后面的循环体，然后进行判断。而while语句要先进行判断，若为真，则执行循环体否则跳出。

### 2.4 for语句

#### 2.4.1 for语句的基本语法

在C语言中，for语句使用最为灵活，它完全可以取代while语句。它的一般形式为：

for(表达式1;表达式2;表达式3) 语句

它的执行过程如下：

1. 先求解表达式1。
2. 求解表达式2，若其值为真(非零)，则执行for语句中指定的内嵌语句，然后执行下面第3)步；若其值为假(0),则结束循环，转到第5)步。
3. 求解表达式3。
4. 转回上面第2)步继续执行。
5. 循环结束，执行for语句下面的一个语句。

for语句最简单的形式也是最容易理解的形式如下：

for(循环变量赋初值；循环条件；循环变量增量) 语句

循环变量赋初值总是一个赋值语句，它用来给循环控制变量赋初值；循环条件是一个关系表达式，它决定什么时候退出循环；循环变量增量，定义循环控制变量每循环一次后按什么方式变化。这三个部分之间用；分开。

例如：

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

{

sum = sum + i;

}

先给i赋初值1，判断i是否小于等于100，若是则执行语句，之后值增加1。再重新判断，直到条件为假，即i>100时，结束循环。

(从汇编的角度来说，for语句要比while语句效率低)

#### 2.4.2 使用for语句时应注意的问题

1. for循环中的“表达式1(循环变量赋初值)”、“表达式2(循环条件)”和“表达式3(循环变量增量)”都是选择项，即可以缺省，但“；”不能缺省。
2. 省略了“表达式1(循环变量赋初值)”，表示不对循环控制变量赋初值。
3. 省略了“表达式2(循环条件)”，则不做其他处理时便成为死循环。(没有判断即永远为真。)

例如：

for(i=1;;i++)

{

sum = sum + i;

}

1. 省略了“表达式3(循环变量增量)”，则不对循环控制变量进行操作，这时可在语句体中加入修改循环控制变量的语句。

例如：

for(i = 1; i <= 100;)

{

sum = sum + i;

i++;

}

1. 省略了“表达式1(循环变量赋初值)”和“表达式3(循环变量增量)”。

例如：

for( ; i <= 100; )

{

sum = sum + i;

i++;

}

1. 三个表达式都可以省略。

例如：

for(;;)语句

相当于：

while(1)语句

是个死循环。

1. 表达式1可以是设置循环变量的初值的赋值表达式，也可以是其他表达式。

例如：

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

sum = sum + i;

1. 表达式1和表达式3可以是一个简单表达式也可以是逗号表达式。

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

{

sum = sum + i;

}

或：

for(i = 0,j = 100; i <= 100; i++,j--)

{

k = i + j;

}

1. 表达式2一般是关系表达式或逻辑表达式，但也可以是数值表达式或字符表达式，只要其值非零，就执行循环体。

例如：

for(i = 0; (c = getchar())!=’\n’; i += c);

又如：

for(;(c = getchar())!=’\n’;)

{

printf(“%c”,c);

}

循环的嵌套

#include <stdio.h>

void main()

{

int i, j, k;

printf("i j k\n");

for(i=0;i<2;i++)

for(j=0;j<2;j++)

for(k=0;k<2;k++)

printf("%d %d %d\n", i, j, k);

}

问题：输出以下图形

\*

\*\*

\*\*\*

\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*\*

程序如下：

#include <stdio.h>

void main()

{

int i, j;

for(i=1;i<=6;i++)

{

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

{

printf("\*");

}

printf("\n");

}

}

重要的编程技巧：调试程序

### 2.5 几种循环的比较

1. 四种循环都可以用来处理同一问题，一般情况下它们可以互相代替。但一般不提倡用goto型循环。
2. 在while循环和do-while循环中，只在while后面的括号内指定循环条件，因此为了使循环能正常结束，应在循环体中包含使循环趋于结束的语句(如i++,或i=i+1等)。

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

1. 用while和do-while循环时，循环变量初始化的操作应在while和do-while语句之前完成而for语句可以在表达式1中实现循环变量的初始化。
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(break语句和continue语句见下节)。而对用goto语句和if语句构成的循环，不能用break语句和continue语句进行控制。**

例题：有1、2、3、4四个数字，能组成多少个互不相同且无重复数字的三位数？都是多少？1.程序分析：可填在百位、十位、个位的数字都是1、2、3、4。组成所有的排列后再去掉不满足条件的排列。

#include <stdio.h>

void main()

{

int i,j,k,a,n = 0;

for(i=1;i<=4;i++)

{

for(j=1;j<=4;j++)

{

for(k=1;k<=4;k++)

{

a =100\*i+10\*j+k;

if (i==j||i==k)

{

continue;

}

if (j==k)

{

continue;

}

printf("%d\n",a);

n++;

}

}

}

printf("共有%d个互不相同且无重复数字的三位数。\n",n);

}

改编的程序：

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

输入0~9之间的一个数字，在0~这个数字之间的所有

数字能组成多少个互不相同且无重复数字的三位数？

都是多少？可以采用如下程序实现。

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <stdio.h>

void main()

{

int i,j,k,a,n = 0;

int m;

printf("请输入数字：\n");

scanf("%d",&m);

if(m>9||m<0)

{

printf("输入错误!!!\n");

return;

}

printf("\n");

for(i=0;i<=m;i++)

{

for(j=0;j<=m;j++)

{

for(k=0;k<=m;k++)

{

a =100\*i+10\*j+k;

if(0==i)

{

continue;

}

if (i==j||i==k)

{

continue;

}

if (j==k)

{

continue;

}

printf("%d ",a);

n++;

}

}

}

printf("\n0~%d这%d个数字共能组成%d个互不相同且无重复数字的三位数。\n",m,m,n);

}

### 2.6 break和continue语句

#### 2.6.1 break语句

break语句可以用来从循环体内跳出循环体，即提前结束循环，接着执行循环下面的语句。一般形式：

break;

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。

注意：

1. break语句对if-else的条件语句不起作用。
2. 在多层循环中，一个break语句只向外跳一层。

示例：

#include <stdio.h>

void main()

{

float area,pi = 3.14159;

int r;

for(r = 1; r <= 10; r++)

{

area = pi\*r\*r;

if(area>100)

{

break;

}

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

}

}

程序的作用是计算r=1到r=10时的圆面积，直到面积area大于100为止。从上面的for循环可以看到：当area》100时，执行break语句，提前结束循环，即不再继续执行其余的几次循环。

break语句的示例：

#include <stdio.h>

#include <conio.h>/\*后面的getch()函数需要该头文件\*/

void main()

{

int i;

char c;

while(1)/\*设置循环\*/

{

i = 0;

c = '\0';/\*变量赋初值\*/

while(c!=13&&c!=27)/\*键盘接收字符直到按回车或ESC键\*/

{

c = getch();/\*getch()函数适用于从控制台读取一个字符，但不显示在屏幕上。

函数原型为 int getch(void)。其返回值为读取的字符。所需的头文件

为conio.h。\*/

printf("%c\n",c);

i++;

}

if (27==c)

{

break;/\*判断若按Esc键则退出循环\*/

}

printf("The No. is %d\n",i-1);

}

printf("The end.\n");

}

#### 2.6.2 continue语句

作用为结束本次循环，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;

#### 2.6.3 break和continue语句的区别

continue语句只结束本次循环，而不是终止整个循环的执行。

break语句则是结束整个循环过程，不再判断执行循环的条件是否成立。

例题：

显示输入的字符，如果按的是Esc键，则退出循环；如果按的是Enter键，则不做任何处理，继续输入下一个字符。

可以采用如下程序实现：

#include <stdio.h>

#include <conio.h>/\*后面的getch函数和putch函数都需要该头文件。\*/

void main()

{

char ch;

for(;;)/\*相当于while(1)永远为真，一直循环下去。\*/

{

ch = getch();/\*字符输入函数。\*/

if(27 == ch)

{

break;/\*按的是Esc键，退出整个循环。\*/

}

if(13 == ch)

{

continue;/\*按的是Enter键，跳过字符输出语句。\*/

}

putch(ch);/\*显示输入的字符。也可以用printf函数打印出来。\*/

}

getch();/\*让程序停一下，按任意键结束。这样可以不让生成的.exe文件一闪而过。可以让程序停下。很好的函数。\*/

}

例题：把100~200之间的不能被3整除的数输出。

可以通过如下程序实现：

#include <stdio.h>

void main()

{

int num;

for(num = 100; num <= 200; num++)

{

if(num%3!=0)

{

printf("%d ",num);

}

}

}

也可以按下面的思路进行：

思路：当n能被3整除时，执行continue语句，结束本次循环(即跳过printf函数语句)，只有n不能被3整除才执行printf函数。

程序如下：

#include <stdio.h>

void main()

{

int num;

for(num = 100; num <= 200; num++)

{

if(0 == num%3)

{

continue;

}

printf("%d ",num);

}

printf("\n");

}

程序举例1：

用≈公式求的近似值，直到某一项的绝对值小于0为止。

程序提示：

1. 要确定计算的精度……

可以配合while循环语句用fabs()函数确定精度来退出。

1. 据观察，分子不变，分母却每次递增2，且正负切换。
2. 记得结果乘以4。

C语言数学函数fabs()的用法：

原型：在VC6.0中原型是double fabs(double x);。

用法：#include <math.h>

功能：求浮点数x的绝对值

说明：计算|x|,当x不为负时返回x，否则返回-x

类似函数abs

#include<stdio.h>

#include<math.h>

void main()

{

int s = 1;

float n = 1.0, t = 1, pi = 0;

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);

}

程序举例2：

求Fibonacci数列前40个数。这个数列有如下特点：第1,2两个数为1,1。从第3个数开始，该数是其前面两个数之和。

即：

F(1) = 1 (n = 1)

F(2) = 1 (n = 2)

… …

F(n) = F(n - 1) + F(n - 2) (n ≥ 3)

可以采用如下程序：

#include <stdio.h>

#include <math.h>

void main()

{

int F[1], n = 1;

F[1] = 1;

F[2] = 1;

while ( n <= 40 )

{

F[n+2] = F[n+1] + F[n];

printf("F[%d] = %d\n", n, F[n]);

n++;

}

}

也可以采用如下程序：

#include <stdio.h>

void main()

{

long int F1, F2, F, n = 1;/\*到后面的值会比较大，所以采用long整型。\*/

F1 = 1;/\*F1代表题目中的F[n-2],

F2代表题目中的F[n-1],

F代表题目中的F。\*/

F2 = 1;

F = 1;

while ( n <= 40 )

{

printf("第%d项 = %d\n", n, F1);/\*这里要输出的值为F1的值,

若输出F则会丢失第一项。\*/

F = F1 + F2;

F1 = F2;/\*把F[n-1]的值覆盖到F[n-2]上，这样被扫描的三个数字才会向后进行。\*/

F2 = F;/\*同理，把F[n]的值覆盖到F[n-1]上,也是为了使被扫描的三个数字向后进行。\*/

n++;

}

}

书上的答案:

#include <stdio.h>

void main()

{

long int F1, F2;

int i;

F1 = 1;

F2 = 1;

for(i = 1; i <= 20; i++)

{

printf("%12ld %12ld ", F1, F2);

if (0 == i % 2)

{

printf("\n");

}

F1 = F1 + F2;

F2 = F2 + F1;

}

}

程序举例3：

写一个程序，允许输入一个数m，并判断m是否为素数？？

可以按下面的程序运行：

#include <stdio.h>

void main()

{

int m, i;

printf("请输入一个整数。\n");

scanf("%d", &m);

for ( i = 2; ; i++)

{

if ( 0 == m % i )

{

printf("%d 是一个合数。\n", m);

break;

}

if( m - 1 == i )

{

printf("%d 是一个素数。\n", m);

break;

}

}

}

也可以用下面的程序搞定：

#include <stdio.h>

#include <math.h>

void main()

{

int m, i;

printf("请输入一个整数。\n");

scanf("%d", &m);

for ( i = 2; i <= sqrt(m); i++ )

{

if ( 0 == m % i )

{

break;

}

}

if ( i > sqrt(m) )

{

printf("%d 是一个素数。\n", m);

}

else

{

printf("%d 是一个合数。\n", m);

}

}

上面的第二个程序会减少循环的次数，因此比第一个程序好一些。

程序举例4：

求100~200之间的所有素数并打印出来。

思路：先会判断一个素数，然后再它的外层加一个循环语句。

#include <stdio.h>

#include <math.h>

void main()

{

int m, i;

for (m = 100; m <= 200; m++)

{

for ( i = 2; i <= sqrt(m); i++ )

{

if ( 0 == m % i )

{

break;

}

}

if ( i > sqrt(m) )

{

printf("%d \n", m);

}

}

}

程序举例5：

译密码！！为使电文保密，往往按一定规律将其转换成密码，收报人在按约定的规律将其译回原文。

例如：可以按以下规律将电文变成密码：

将字母A变成字母E，a变成e，即变成其后的第4个字母，W变成A，X变成B,Y变成C，Z变成D。

#include <stdio.h>

void main()

{

char ch = '\0';

while (ch != '\n')

{

ch = getchar();

if ( ( 'a' <= ch && 'w' > ch ) || ( 'A' <= ch && 'W' > ch ) )

{

putchar(ch + 4);

}

if ( ( 'w' <= ch && 'z' >= ch) || ( 'W' <= ch && 'Z' >= ch ) )

{

putchar(ch - 22);

}

}

printf("\n");

}

带有加密和解密的两种功能的程序如下：

#include <stdio.h>

#include <conio.h>

void main()

{

char ch = '\0';

char a;

char c;

printf("0退出程序\n1由原文到密文的加密过程\n2由密文到原文的解密过程\n\n");

while (1)

{

printf("请输入0或1或2进行选择\n\n");

a = getch();

putch(a);

printf("\n");

if ( '0' != a && '1' != a && '2' != a)

{

printf("输入错误\n");

}

if ( '0' == a)

{

break;

}

if ( '1' == a)

{

printf("加密\n");

}

if ( '2' == a)

{

printf("解密\n");

}

while (ch != '\n' && '2' == a)

{

ch = getchar();

if ('A' > ch || ch > 'z')

{

putchar(ch);

}

if ( 'Z' < ch && ch < 'a' )

{

putchar(ch);

}

if ( ( 'a' <= ch && 'w' > ch ) || ( 'A' <= ch && 'W' > ch ) )

{

putchar(ch + 4);

}

if ( ( 'w' <= ch && 'z' >= ch) || ( 'W' <= ch && 'Z' >= ch ) )

{

putchar(ch - 22);

}

}

while (ch != '\n' && '1' == a)

{

ch = getchar();

if ('A' > ch || ch > 'z')

{

putchar(ch);

}

if ( 'Z' < ch && ch < 'a' )

{

putchar(ch);

}

if ( ( 'e' < ch && 'z' >= ch ) || ( 'E' < ch && 'Z' >= ch ) )

{

putchar(ch - 4);

}

if ( ( 'a' <= ch && 'e' >= ch) || ( 'A' <= ch && 'E' >= ch ) )

{

putchar(ch + 22);

}

}

ch = '\0';

printf("\n");

}

}

# 第五章 数组

## 1 数组的概念

具有相同的数据类型。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。

概念：

在程序设计中，为了处理方便，把具有相同类型的若干变量按有序的形式组织起来。这些按序排列的同类数据元素的集合称为数组。在C语言中，数组属于构造数据类型。一个数组可以分解为多个数组元素，这些数组元素可以是基本数据类型或是构造类型。因此按数组元素的类型不同，数组又可分为数值数组、字符数组、指针数组、结构数组等各种类别。

数组：具有相同类型的数据组成的序列，是有序集合。

数组中的每一个数据称：1.数组元素 2.下标变量

数组元素：由其所在的位置序号(称数组元素的下标)来区分。

用数组名与下标 可以用统一的方式来处理数组中的所有元素，从而方便的实现处理一批具有相同性质数据的问题。

注意：数组元素有序不是指元素大小顺序

## 2 一维数组的定义和引用

一维数组的定义方式：

在C语言中使用数组必须先进行定义。

一维数组的定义方式为：

类型说明符 数组名[常量表达式]

例如：int a[10];

它表示定义了一个整型数组，数组名为a，此数组有10个元素，10个元素都是整型变量！

需要注意的是：

* 1. 类型说明符是任一种基本数据类型或构造数据类型。对于同一个数组，其所有元素的数据类型都是相同的。
  2. 数组名是用户定义的数组标识符。书写规则应符合标识符的书写规定。
  3. 方括号中的常量表达式表示数据元素的个数，也称为数组的长度。
  4. 允许在同一个类型说明符中，说明多个数组和多个变量。例如：int a, b, c, d, k1[10], k2[20];
  5. a[10],表示a数组有10个元素，注意下标是从0开始的这10个元素是a[0],a[1],a[2]……a[9],这10元素。特别注意，按上面的定义，不存在数组元素a[10]。
  6. C语言不允许对数组的大小作动态定义，即数组的大小不依赖于程序运行过程中变量的值。例如，下面这样定义数组是不行的：

int n;

scanf(“%d”,&n);/\*在程序中临时输入数组的大小\*/

int a [n];

一些常见的错误：

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的大小/

以下是正确的演示：

int a[10];//说明整型数组a，有10个元素。

float b[10], c[20];//说明实型数组b，有10个元素，实型数组c，由0个元素。

char ch[20];//说明字符数组ch，有20个元素。

重点补充：一维数组在内存中的存放

一维数组： int mark[100];

|  |
| --- |
| 86 |
| 92 |
| 77 |
| 52 |
| … |
| 94 |

mark[0] 低地址

mark[1]

mark[2]

mark[3]

……

mark[99] 高地址

每个数据元素占用的字节数，就是基类型的字节数，一个元素占4个字节。

## 3 一维数组的引用

数组元素是组成数组的基本单元。数组元素也是一种变量，其标识方法为数组名后跟一个下标。下标表示了元素在数组中的标号。

数组元素的一般形式为：

数组名[下标]

(下标可以是整型常量或整型表达式。)

例如：

a[0] = a[5] + a[7] – a[2\*3]

a[i+j]

a[i++]

都是合法的数组元素。

注意事项：

数组元素通常也称为下标变量。必须先定义数组，才能使用下标变量。在C语言中只能逐个地使用下表变量，而不能一次引用整个数组。

例如：输出10个元素的数组必须使用循环语句逐个输出个下标变量：

for (i = 0; i < 10; i++)

{

printf(“%d”,a[i]);

}

而不能用一个语句输出整个数组。

下面的写法数错误的：

printf(“%d”, a);

定义数组时用到的”数组名[常量表达式]”和引用数组元素时用到的“数组名[下标]”是有区别的。

例如：int a[10];/\*定义数组长度为10\*/

t = a[6];/\*引用a数组中序号为6的元素。此时6不代表数组长度。\*/

示例：

#include <stdio.h>

int main(void)

{

int i, a[10];

for (i = 0; i<=9; i++)

{

a[i] = i;

}

for (i = 9; i>=0; i--)

{

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

}

return 0;

}

## 4 一维数组的初始化

小秘密：给数组赋值的方法除了用赋值语句对数组元素逐个赋值外，还可以采用初始化赋值和动态赋值的方法。

数组初始化赋值是指在数组定义时给数组元素赋予初值。数组初始化时在编译阶段进行的。这样将减少运行时间，提高效率。

注意：之前用赋值语句或输入语句也可以给数组元素指定初值，是在运行时完成。

初始化赋值的一般形式为：

类型说明符 数组名[常量表达式]={值，值，……值}；

### 4.1 数组初始化的实现方法

1.在定义数组时对数组元素赋以初值。例如：

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

将数组元素的初值依次放在一对花括号内。经过上面的定义和初始化之后，a[0]= 0; a[1]= 1; a[2]= 2; a[3]= 3; a[4]= 4; a[5]= 5; a[6]= 6; a[7]= 7; a[8]= 8; a[9]= 9.

(数组名就是数组的首地址！！)

2.也可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，但花括号内值提供5个初值，这表示只给钱面5个元素赋初值，后5个元素值为0.

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};或int a[10] = {0};

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

例如：

int a[5] = {0, 1, 2, 3, 4, 5};

也可以写成：int a[] = {0, 1, 2, 3, 4, 5};

5.数组初始化与未初始化比较

6.动态赋值的方法：

#include <stdio.h>

int main(void)

{

int i, a[10], max;

printf("input ten numbers:\n");

for(i = 0; i <= 9; i++)

{

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

}

max = a[0];

for(i = 0; i <= 9; i++)

{

max = (a[i] > max) ? a[i]:max;

}

printf("%d\n", max);

return 0;

}

/\*用户输入10个整数，然后输出其中最大数。\*/

### 4.2 冒泡排序法(起泡法)

#### 4.2.1 代码

对10个书排序(由小到大)冒泡法的思路是：将相邻两个数比较，将小的调到前头。

#include <stdio.h>

int main(void)

{

int a[10];

int i, j, k;

printf("Please input ten numbers.\n");

for (i = 0; i <= 9; i++)

{

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

}

for(j = 0; j <= 9; j++)

{

for (k = 0; k < 9 - j; k++)

{

if ( a[k] >= a[k+1] )

{

i = a[k];

a[k] = a[k+1];

a[k+1] = i;

}

}

}

for (i = 0; i <= 9; i++)

{

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

}

printf("\n");

return 0;

}

该排序方法很重要。

#### 4.2.2 过程提示

1.先通过动态赋值的方法让用户输入10个待排序的数字，通过数组保存起来。

2.掌握两个循环，外层循环限制有多少趟比较，内层循环限制某趟比较要进行两两对比的次数。

3.如果a[i] > a[i+1],则将两数进行交换，因为要求从小到大排列嘛~

4.把最终的结果输出到屏幕！

5.完成！！

## 5 二维数组的定义和引用

### 5.1 二维数组的定义

二维数组定义的一般形式为：

类型说明符 [常量表达式] [常量表达式]；

例如：定义a 为(3行4列)的数组，b为(5行10列)的数组。如下：

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

不能写成 float a[3,4], b[5,10];

这里的3行4列在内存中是怎样表示的呢？

例如：

int a[3][4];

说明了一个三行四列的数组，数组名为a，其下标变量的类型为整数。该数组的下标变量共有个，即：

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

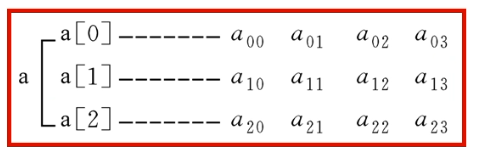
a[1][0], a[1][1], a[1][2], a[1][3]

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

第一个常量表达式代表行数，第二个常量表达式代表列数。

我们可以把二维数组看做是一种特殊的一维数组：它的元素又是一个一维数组。

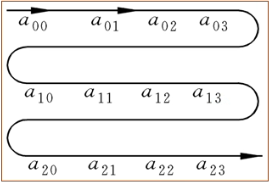
例如，可以把a看做是一个一维数组，它有3个元素：a[0],a[1],a[2]，每个元素又是一个包含4个元素的一维数组。



二维数组在概念上是二维的，但是，实际的硬件存储器却是连续编址的，也就是说存储器单元是按一维线性排列的。

如何在一维存储器中存放二维数组，可有两种方式：一种是按行排列，即放完一行之后顺次放入第二行。另一种是按列排列，即放完一列之后再顺次放入第二列。在C语言中，二维数组是按行排列的。

下图表示a[3][4]在内存中的存放方式：

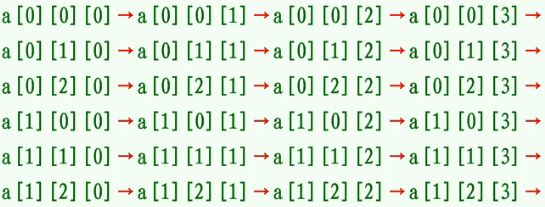


衍生：多维数组的定义

有了二维数组的基础，那么多维数组如何定义呢?

定义三维数组：float a[2][3][4];

多维数组元素在内存中的排列顺序：第一维的下标变化最慢，最右边的下标变化最快。



### 5.2 二维数组的引用和初始化

数据类型 数组名[常量表达式1][常量表达式2]={初始化数据}；

我们有4种方法对二维数组初始化：

(1)直接分行给二维数组赋初值。如：

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

(2)可以将所有数据写在一个花括号内，按数组排列的顺序对个元素赋初值。如：

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

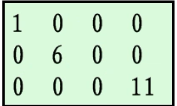
(3)可以对部分元素赋初值。如：

int a[3][4] = {{1},{5},{9}};//这样初始化的是每一行的第一个元素!!

int a[3][4] = {1, 5, 9};//这样初始化的是从第一行开始的三个元素!!

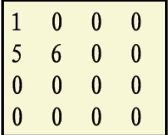
也可以对各行中的某一元素赋初值，如：

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



其实也可以只对某几行元素赋初值。如：

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



小技巧：在重复注释的时候。

比如：

//int a[3][4] = {{1},{5},{9}};//这样初始化。

这样重复注释可能会引起一些不必要的麻烦。因为在C语言中是不能嵌套注释的。

解决此问题的方法是采用宏注释。当然：宏注释的话不会出现红色的字体。方法如下：

#if( 0 )

int a[3][4] = {{1},{5},{9}};//这样初始化。

#endif

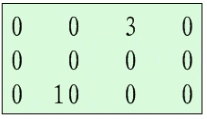
(4)如果对全部元素都赋初值，则定义数组是对第一维的长度可以不指定，但第二维的长度不能省。如：

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

它等价于：

int a[][4] = {1,2,3,4,5,6,7,8,9,10,11,12};//不过没有必要的。

在定义是也可以只对部分元素赋初值而省略第一维的长度，但应分行赋初值。如：

int a[][4] = {{0, 0, 3}, {}, {0, 10}};//也是没有必要的。

例题：如图，一个学习小组有5人，每个人有三门课的考试成绩。将各个数据保存到二维数组a[5][3]中，并求劝阻分科的平均成绩和总平均成绩。



对数组a[5][3]:

1)按行分段赋值可写为(首选)：

int a[5][3] = {{80, 75, 92}, {61, 65, 71}, {59, 63, 70}, {85, 87,90}, {76, 77, 85}};

2)按行连续赋值可写为：

int a[5][3] = {80, 75, 92, 61, 65, 71, 59, 63, 70, 85, 87, 90, 76, 77, 85};

这两种赋初值的结果是完全相同的。

#include <stdio.h>

int main(void)

{

int a[5][3] = {{80, 75, 92}, {61, 65, 71}, {59, 63, 70}, {85, 87, 90}, {76, 77, 85}};

int i, j, sum = 0, average, v[3];

for (i = 0; i < 3; i++)//表示科目

{

for (j = 0; j < 5; j++)

{

sum += a[j][i];

}

v[i] = sum / 5;

sum = 0;

}

average = (v[0] + v[1] + v[2]) / 3;

printf("math : %d\nc language:%d\ndFoxpro:%d\n", v[0], v[1], v[2]);

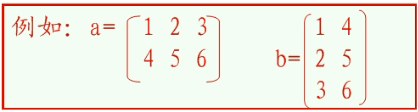
printf("total:%d\n", average);

return 0;

}

示例：

将一个二维数组行和列元素互换，存到另一个二维数组中。

i.e.将数组a[2][3]转化为数组b[3][2]。

#include <stdio.h>

int main(void)

{

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

int b[3][2] = {0};

int i, j;

for ( i = 0; i < 2; i++)

{

for ( j = 0; j < 3; j++)

{

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

}

printf("\n");

}

printf("\n");

for (i = 0; i < 2; i++)

{

for (j = 0; j < 3; j++)

{

b[j][i] = a[i][j];

}

}

for ( j = 0; j < 3; j++)

{

for ( i = 0; i < 2; i++)

{

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

}

printf("\n");

}

return 0;

}

思路提示：

化繁为简：无论多少维数组，分解成一维数组进行转换！

示例：

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

可以按以下程序实现：

#include <stdio.h>

int main(void)

{

int a[3][4];

int i, j, m, k, max;

printf("Please input 12 numbers.\n");

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

{

for ( j = 0; j < 4; j++)

{

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

}

}

printf("\n");

max = a[0][0] > a[0][1] ? a[0][0] : a[0][1];

m = 0;

k = a[0][0] > a[0][1] ? 0 : 1;

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

{

for ( j = 0; j < 3; j++)

{

max = ( max > a[i][j+1] ) ? max : a[i][j+1];

m = ( max > a[i][j+1] ) ? m : i;

k = ( max > a[i][j+1] ) ? k : (j+1);

}

}

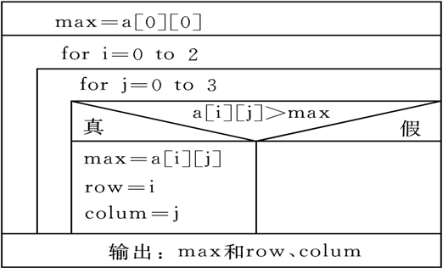
printf("a[%d][%d] = %d\n", m, k, max);

return 0;

}

还可以按如下思路进行编程：

思路：



这就是ms图。

程序如下：

#include <stdio.h>

int main(void)

{

int i, j, row = 0, colum = 0, max;

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

max = a[0][0];

for (i = 0; i <= 2; i++)

{

for(j = 0; j <= 3; j++)

{

if(a[i][j] > max)

{

max = a[i][j];

row = i;

colum = j;

}

}

}

printf("max = a[%d][%d] = %d\n", row, colum, max);

return 0;

}

示例：

从键盘上输入9个整数，(对照九宫格的形式，输入三行，每行输入三个数)保存在二维数组中，按数组原来的位置输出第一行和第一列的所有元素。

#include <stdio.h>

int main(void)

{

int a[3][3];

int i, j, m, k, max;

printf("Please input 9 numbers.\n");

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

{

for ( j = 0; j < 3; j++)

{

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

}

}

printf("\nYou just inputted these numbers as follow form.\n");

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

{

for (j = 0; j < 3; j++)

{

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

}

printf("\n");

}

printf("\nAnd following form is the one you want.\n");

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

{

for (j = 0; j < 3; j++)

{

if (1 == i || 1 == j)

{

printf("%d ", a[i][j]);//也可以这样编写printf("%-6d", a[i][j]);

}

else

{

printf(" ");//也可以这样编写printf("%-6c", ' ');

}

}

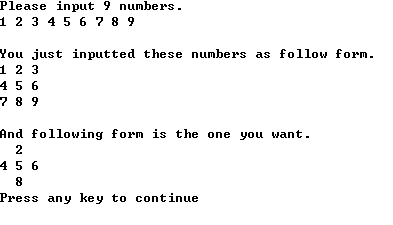
printf("\n");

}

return 0;

}

上面的程序中，如果你输入1 2 3 4 5 6 7 8 9会输出：



(注意：外层循环是行循环，内层循环是列循环。)

### 5.3 其应用(二分法)

示例： 二分法举例

(利用数组进行数据查找——(二分法)折半查找法介绍！！)

适应情况：在一批有序数组中查找某数

基本思想：选定这批数中居中间位置的一个数与所查数比较，看是否为所找之数，若不是，利用数据的有序性，可以决定所找的数是在选定数之前还是之后，从而很快可以将查找范围缩小一半。以同样的方法在选定区域中进行查找，每次都会讲查找范围缩小一半，从而很快地找到目的数。

假设在数组a中的数据是按由小到大的顺序排列的：-12 0 6 16 23 56 80 100 110 115，从键盘上输入一个数，判定该数是否在数组中，若在，输出所在序号。

可以用下面的程序实现：

#include <stdio.h>

int main(void)

{

int a[] = {-12, 0, 6, 16, 23, 56, 80, 100, 110, 115};

int num, i;

printf("Please input a number.\n");

scanf("%d", &num);

for ( i = 0; i <= 9; i++)

{

if ( num == a[i] )

{

break;

}

}

if ( num == a[i] )

{

printf("a[%d] = num = %d\n", i, num);

}

else

{

printf("There is no such a number in the numbers group.\n");

}

return 0;

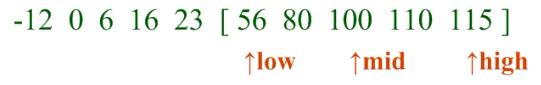
}

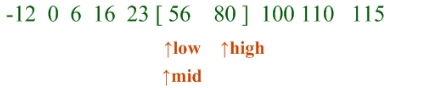
也可以用下面的程序实现：

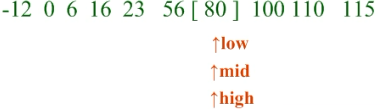
其思路是这样的：

第一步：假设要找的是80。设low，mid和high三个变量，分别指示数列中的起始元素、中间元素与最后元素的位置，其初始化为low = 0，high = 9，mid = 4，判断mid指示的是否为所求，mid指示的数是23，不是要找的80，须继续进行查找。 

第二步：确定新的查找区间。因为80大于23，所以查找范围可以缩小为23后面的数，新的查找区间为[56 80 100 110 115],low,mid,high分别指向新区间的开始、中间与最后一个数。实际上high不变，将low(low = mid + 1)指向56，mid(mid = (low + high) / 2 )指向100，还不是要找的80 ，仍需继续查找。



第三步：上一步中，所找数80比mid指示的100要小，可知新的查找区间为[56 80]，low不变，mid与high的值作相应的修改。mid指示的数为56，还要继续查找。

第四步：根据上一步的结果，80大于mid指示的数56，可以确定新的查找区间为[80],此时，low与high都指向80，mid亦指向80，即找到了80，到此为止，查找过程完成。 

注意：若在查找过程中，出现low>high的情况，则说明，序列中没有该数，亦结束查找过程。

其实现方法如下：

#include <stdio.h>

int main (void)

{

int a[10] = {-12, 0, 6, 16, 23, 56, 80, 100, 110, 115};

int low, mid, high;

int number;

printf("Please input the number that you want to find.\n");

scanf("%d", &number);

for (low = 0, mid = 4, high = 9; ; )

{

if (a[mid] == number)

{

printf("a[%d] = %d is the number that is in this group.\n", mid, a[mid]);

break;

}

if (a[high] == number)

{

printf("a[%d] = %d is the number that is in this group.\n", high, a[high]);

break;

}

if (a[low] == number)

{

printf("a[%d] = %d is the number that is in this group.\n", low, a[low]);

break;

}

if (a[mid] < number)

{

low = mid;

mid = (high + low) / 2;

}

else

{

high = mid;

mid = (high + low) / 2;

}

if ((mid == high) || (mid == low))

{

printf("There is no such a number like %d in the group.\n", number);

break;

}

}

return 0;

}

标准方法:

#include <stdio.h>

#define M 10/\*这里千万不要加分号!!!\*/

int main (void)

{

static int a[M] = {-12, 0, 6, 16, 23, 56, 80, 100, 110, 115};

/\*一个程序在编译运行的时候，普通变量时存放在栈区，我们还会学习堆，

而这个static会使得变量存放在data区。无论任何语言，整个内存主要分为四大区：

CODE，DATA（数据），STACK（栈），HEAP\*/

int n, low, mid, high, found;

low = 0;

high = M-1;

found = 0;

printf("Input a number to be searched:\n");

do

{

scanf("%d", &n);

getchar();/\*在流上获取一个字符，但是不把获取的值赋给变量，

只是凭空的读取到堆栈中。调用完就删掉。它的作用是消耗流一个字符。\*/

}while( n > a[M-1] || n < a[0] );/\*do while语句的分号不可少。\*/

/\*该步骤是为了1.增加效率，提高速度。2.对输入的数据进行筛选，阻挡非法数据。\*/

while (low <= high)

{

mid = (low + high) / 2;

if (n == a[mid])

{

found = 1;

break;

}/\*找到，结束循环。\*/

else if (n > a[mid])

{

low = mid + 1;

}

else

{

high = mid - 1;

}

}

if (1 == found)

{

printf("The index of %d is %d.\n", n, mid);

}

else

{

printf("There is not %d.\n", n);

}

return 0;

}

其改进形式如下：

#include <stdio.h>

#define M 10/\*这里千万不要加分号!!!\*/

int main (void)

{

static int a[M] = {-12, 0, 6, 16, 23, 56, 80, 100, 110, 115};

/\*一个程序在编译运行的时候，普通变量时存放在栈区，我们还会学习堆，

而这个static会使得变量存放在data区。无论任何语言，整个内存主要分为四大区：

CODE，DATA（数据），STACK（栈），HEAP\*/

int d, n, low, mid, high, found;

low = 0;

high = M-1;

found = 0;

printf("Input a number to be searched:\n");

while (scanf("%d", &n) != 1)

{

printf("Illegal input!!!\nPlease input again!!!\n");

getchar();

}

/\*scanf函数的返回值是成功读入的数量。如：如果语句是这样写的scanf("%d %d", &n, &m);，

则函数成功读入数据后的返回值是2；如果只有m被成功读入，则返回值也是1；如果都没有被成功读入，

则返回值是0；如果遇到错误或遇到end of file，返回值为EOF。这里是运用了scanf函数的返回值来进行

对非法数据输入的筛选。即若返回值是1，则不进行重新输入，若返回值不是1，则进行重新输入。

getchar函数的作用也是和上面的程序的一样。\*/

while (low <= high)

{

mid = (low + high) / 2;

if (n == a[mid])

{

found = 1;

break;

}/\*找到，结束循环。\*/

else if (n > a[mid])

{

low = mid + 1;

}

else

{

high = mid - 1;

}

}

if (1 == found)

{

printf("The index of %d is %d.\n", n, mid);

}

else

{

printf("There is not %d.\n", n);

}

return 0;

}

/\*但是改程序还存在不好的地方就是在于：当用户输入若干个非法的字符时，会出现若干个非法提示。\*/

# 第六章 函数

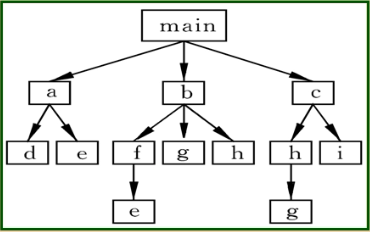
## 1 概述

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语言可由一个主函数和若干个其它函数构成。

## 2 函数的调用关系

****

由主函数调用其它函数，其它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。

示例：

#include <stdio.h>

int main(void)

{

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\_nessage 函数\*/

printstar();/\*调用printstar 函数\*/

return 0;

}

void printstar()/\*定义printstar函数\*/

{

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

}

void print\_message()/\*定义print\_message函数\*/

{

printf("How do you do!\n");

}

说明：

(1)一个C程序由一个或多个程序模块组成，每一个程序模块作为一个源程序文件。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程序执行时从main函数开始的，如是在main函数中调用其它函数，在调用后流程返回到main函数，在main函数中结束整个程序的运行。

(4)所有的函数都是平行的，即在定义函数时是分别进行的，是相互独立的。一个函数并不从属另一个函数，即函数不能嵌套定义。函数间可以互相调用，但不能调用main函数。main函数时系统调用的。

(5)从用户使用的角度来看，函数用两种：

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。应该说明，不同的C系统提供的库函数的数量和功能会有一些不同，当然许多基本的函数是共同的。

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：

无参函数。如示例中的printstar和print\_message函数就是无参函数。在调用无参函数时，主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：上面程序中的printstar函数。

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

定义无参数的一般形式为：

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。

示例中的printstar和print\_message函数为void类型，表示不需要带回函数值。

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 x, 和 int y 是形参。\*/

{

int z;/\*函数体中的声明部分\*/

z = x > y? x : y;

return (z);

}

定义空函数的一般形式为：

类型标识符 函数名()

{}

例如：

dummy()

{}

调用此函数时，什么工作也不做，没有任何实际作用。在主调函数中写上“dummy()；”表明“这里要调用一个函数”，而现在这个函数没有起作用，等以后扩充函数功能时补充上。

## 3 函数参数和函数的值

### 3.1 形式参数和实际参数

在前面提到的有参函数中，在定义函数时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”)。

大多数情况下，主调函数和被调用函数之间有数据传递的关系。

return后面的括弧中的值()作为函数带回的值(称函数返回值)。

**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**

示例：

#include <stdio.h>

void main()

{

int max (int x,int y);//形参

int a, b, c;

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

c = max (a, b);

printf("Max is %d.\n", c);

}

int max (int x, int y)//形参

{

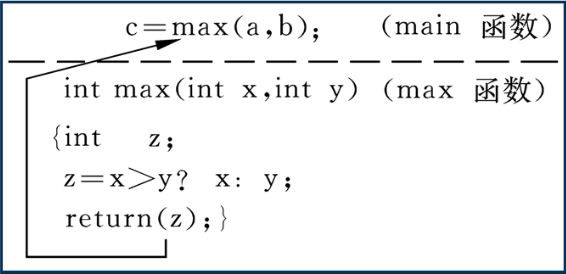
int z;

z = x > y ? x : y;

return (z);

}

通过函数调用，使两个函数中的数据发生联系



关于形参和实参的一些重要说明：

(1)在定义函数中指定的形参，在未出现函数调用时，它们并不占内存中的存储单元。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。

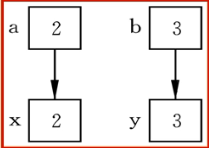
(2)实参可以是常量、变量或表达式，如：max (3,a+b);但要求它们有确定的值。在调用时将实参的值赋给形参。

(3)在被定义的函数中，必须指定形参的类型(示例中的“c = max(a,b);”)。

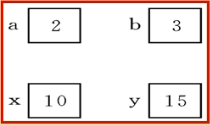
(4)实参与形参的类型应该相同或复制兼容。示例中的实参和形参都是整型。如果实参微整形而形参x为实型，或者相反，则按第三章介绍的不同类型数值的赋值规则进行转换。

例如：实参值a为3.5，而形参x为整型，则将实数3.5转换成整数3，然后送到形参b。字符型与整型可以互相通用。

(5)在C语言中，实参向对形参的数据传递是“值传递”(相当于copy)，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。



在调用函数的时候，给形参分配存储单元，并将实参对应的值传递给形参，调用结束后，形参单元被释放，实参单元仍将保留并维持原值。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。



### 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：示例中，max(2,3)的值是3，max(5,2)的值是5。赋值语句将这个函数值赋给变量C。

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：

return语句将被调用函数中的一个确定值带回主调函数中去。

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

一个函数可以有一个以上的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 2;”等价于“return (2);”

return后面的值可以是一个表达式(只要这个表大会最后会得到一个确定的值就可以)。如：

int max (int x, int y)

{

return (x>y?x:y);

}

(2)函数的返回值应当属于某一个确定的类型，在定义函数时指定函数返回值的类型。

例如：下面是3个函数的首行：

int max (float x, float y)/\*函数值为整型\*/

char letter (char c1, char c2)/\*函数值为字符型\*/

double min (int x, int y)/\*函数值为双精度型\*/

特别提示：

在C语言中，凡是不加类型说明的函数，自动按整型处理。

在示例中的max函数首行的函数类型int可以省写，用Turbo C 2.0编译程序是能通过，单用Turbo C++ 3.0编译程序时不能通过，应为C++要求所有函数都必须指定函数类型。

因此，建议在定义时对所有函数都指定函数类型。

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

如果函数值的类型和return语句中表达式的值不一致，则以函数类型为准。对数值型数据，可以自动进行类型转换。即函数类型决定返回值的类型。

(4)对于不带回值的函数，应当用“void”定义函数为“无类型”(或称“空类型”)。

这样，系统就保证不使用函数带回任何值，即禁止在调用函数中使用被调函数的返回值。

此时在函数体中不得出现return语句。

示例：

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

return (z);

}

## 4 函数的调用

### 4.1 函数调用的一般形式

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。

如果实参表列包含多个实参，则个参数间用逗号隔开。实参与形参的个数应相等，类型应匹配。实参与形参按顺序对应，一一传递数据。

注意：

如果实参表列包括多个实参，对实参求值的顺序并不是确定的，有的系统按自左至右顺序求实参的值，有的系统则按自右至左顺序。

许多C版本是按自右而左的顺序求值，例如Turbo C++。

验证是自左至右还是自右至左的方法。一种是\_cdecl方法调用，是标准的C调用方法，自右至左，需要自己清空栈。另一种是stdcall方法，是windows程序所支持的，也是C和C++新标准的调用方法，自右至左，由编译器自动清空栈。

示例：

#include<stdio.h>

void main()

{

int f(int a,int b);

int i = 2, p;

p = f(i, ++i);

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

}

int f(int a, int b)

{

int c;

if(a > b)

{

c = 1;

}

else if (a == b)

{

c = 0;

}

else

c = -1;

return (c);

}

对于函数调用

int i = 2, p;

p = f(i, ++i);

如果按自左至右顺序求实参的值，则函数调用相当于f(2, 3)。

如果按自右至左顺序求实参的值，则函数调用相当于f(3, 3)。

上面程序的输出结果是0，说明：只有两个数字相等的情况才是这样的结果，所以是自右至左。

### 4.2 函数的调用方式

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：

#### 4.2.1 函数语句

把函数调用作为一个语句。如示例中的printstar(),这时不要求函数带返回值，只要求函数完成一定的操作。

#### 4.2.2 函数表达式

函数出现在一个表达式中，这种表达式称为函数表达式。这时要求函数带回一个确定的值以参加表达式的运算。例如：c = 2 \* max(a, b);

#### 4.2.3 函数参数

函数调用作为一个函数的实参。例如：

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

其中max(b, c)是一次函数调用，它的值作为max另一次调用的实参。m的值是a、b、c三者中的最大者。又如：printf(“%d”, max(a, b));也是把max(a, b)作为printf函数的一个参数。

函数调用作为函数的参数，实质上也是函数表达式形式调用的一种，因为函数的参数本来就要求是表达式形式。

### 4.3 对被调用函数的声明和函数原型

#### 4.3.1 调用另一个函数的条件

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命令将调用有关库函数时所需用到的信息“包含”到本文件中来。

(3)如果使用用户自己定义的函数，而该函数的位置在调用它的函数(即主调函数)的后面(在同一个文件中)，应该在主调函数中对被调用的函数作声明。

#### 4.3.2 区别关键词

“声明”一词的原文是delaration，过去在许多书中把它译为“说明”。声明的作用是把函数名、函数参数的个数和参数类型等信息通知编译系统，以便在遇到函数时，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)。

例如刚才的程序中声明函数int f(int a,int b);也可以写成int f(int, int);这说明编译器只要求知道函数的参数个数以及类型，不必管它的形参的名字。但是我们在变成中尽量把名字写全，这样会更有章法。

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，包括制定函数名、函数值类型、形参及其类型、函数体等，它是一个完整的、独立的函数单位。(占内存)

而函数的声明的作用则是把函数的名字、函数类型以及形参的类型、个数和顺序通知编译系统，以便在调用该函数是系统按此进行对照检查。(不占内存)

示例：求浮点型数据的和。

#include <stdio.h>

void main()

{

float add(float x, float y);/\*对函数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;

z = x + y;/\*函数体\*/

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，会根据函数搜捕提供的信息对函数的调用作正确性检查。

函数double pow(double x, double y);功能：求x的y次幂。

例：

/\*\*\*\*统计从键盘上输入的字符数，空格数，回车数，换行数。

\*\*\*/

#include <stdio.h>

void main()

{

int c, nb, nt, nl;

nb = nt = nl = 0;

while ( (c = getchar()) != EOF )/\*EOF相当于Ctrl+Z键，或F6键\*/

{

if (c == ' ')

++nb;

if (c == '\t')

++nt;

if (c == '\n')

++nl;

}

printf("%d %d %d\n", nb, nt, nl);

}

## 5 函数的嵌套调用

### 5.1 函数调用的释义

嵌套调用就是在定义一个函数时，其函数体内又包含另一函数的完整定义。

然而，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，有调用另一个函数。

例子：

main()

{

a();

}

a()

{

b();

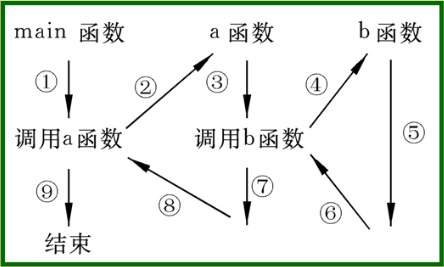
}

b()

{

return;

}



例题：

计算(“！”在高等数学里是阶乘的意思，)

思路：

可以编写两个函数，一个是用来计算平方值的函数square，另一个是用来计算阶乘值的函数factorial。

主函数先调用square计算出平方值，再在square中以平方值为实参，调用factorial计算其阶乘值，然后返回square，再返回主函数，在循环程序中计算累加和。

#include <stdio.h>

int main(void)

{

int factorial(int x);/\*声明阶乘函数\*/

int square(int x);/\*声明平方函数\*/

int a = 2, b = 3;

int s;

s = factorial( square(a) ) + factorial( square(b) );/\*调用两个函数并嵌套使用\*/

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

return 0;

}

int factorial(int x)/\*定义阶乘函数\*/

{

int z = x;

while(--x)

{

z \*= x;

}

return z;

}

int square(int x)/\*定义平方函数\*/

{

int z;

z = x \* x;

return z;

}

标准答案：

#include <stdio.h>

long square(int p); //实现平方

long factorial(int q); //实现阶乘

void main()

{

int i;

long s = 0;

for (i = 2; i <= 3; i++)

{

s = s + square(i);//square函数已经是返回一个数阶乘后的结果了。

}

printf("%ld\n", s);

}

long square(int p)

{

int k;

long r;

long factorial(int);

k = p \* p;

r = factorial(k);

return r;

}

long factorial(int q)

{

long c = 1;

int i;

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

{

c \*= i;

}

return c;

}

## 6 递归(属于算法的知识)

在调用一个函数的过程中又出现直接调用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。

例如：

int f(int x)

{

int y, z;

z = f(y);

return (2\*z);

}

在递归中必须要有一个退出的条件！

例题：

求也可以用递归的方法，即等于，而，……。可用下面的递归公式表示：

n=1 (n=0,1)

n\*(n-1)! (n > 1)

#include <stdio.h>

long factorial(int q);

void main()

{

int i;

long s;

printf("input a integer number:\n");

scanf("%d", &i);

s = factorial(i);

printf("%ld\n", s);

}

long factorial(int q)

{

long c;

if (q < 0)

{

printf("q < 0, input error!\n");

}

else if(0 == q || 1 == q)//退出的条件

{

c = 1;

}

else

{

c = factorial(q - 1) \* q;

}

return c;

}

程序详解：

程序中给出的函数factorial()是一个递归函数。主函数调用factorial()后即进入函数factorial()执行。

如果n < 0, n == 0 或n == 1时都将结束函数的执行，否则就递归调用factorial()函数自身。

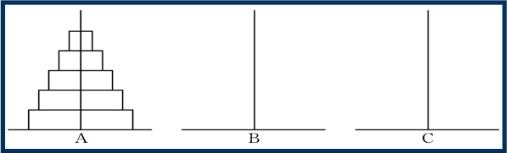
由于每次递归调用的实参为n-1，即把n-1的值赋予形参n，最后当n-1的值为1时再作递归调用，形参n的值也为1，将使递归终止。然后可逐层退回。

递归算法是效率低下的算法！！

有些问题则只能用递归算法才能实现。典型的问题是Hanoi塔问题。

例题：

Hanoi塔问题。这是一个古典的数学问题，是一个用递归方法阶梯的典型例子。问题是这样的：古代有一个梵塔，塔内有3座A、B、C，开始时A座上有64个盘子，盘子大小不等，大的在下，小的在上(见图)



有一个老和尚想把这64个盘子从A座移到C座，但每次只允许移动一个盘，且在移动过程中在3个座上都始终保持大盘在下，小盘在上。在移动过程中可以利用B座，要求编程序打印出移动的步骤。

为了便于理解，我们先分析将A座上3个盘子移到C座的过程：

(1)将A座上2个盘子移动到B座上(借助C)；

(2)将A座上1个盘子移动到C座上；

(3)将B座上2个盘子移动到C座上(借助A)。

其中第(2)步可以直接实现。第(1)步又可以用递归方法分解为：

1.1将A上1个盘子从A移动到C；

1.2将A上1个盘子从A移动到B；

1.3将C上1个盘子从C移动到B。

第(3)步可以分解为：

3.1将B上1个盘子从B移动到A上；

3.2将B上1个盘子从B移动到C上；

3.3将A上1个盘子从A移动到C上。

将以上综合起来，可得到移动3个盘子的步骤为A→C,A→B,C→B,A→C,B→A,B→C,A→C。

由上面的分析可知：将n个盘子从A座移动到C座可以分解为以下3个步骤：

(1)将A上n-1个盘子借助C座先移到B座上。

(2)把A座上剩下的一个盘移到C座上。

(3)将n-1个盘从B座借助A座移到C座上。

标准答案：

#include <stdio.h>

void Hanoi(int n, char A, char B, char C)

/\*定义Hanoi函数，将n个盘子从A座借助B座，移动到C座\*/

{

void move(char x, char y);

/\*对move函数进行声明\*/

if (1 == n)

{

move(A, C);

}

else

{

Hanoi(n-1, A, B, C);

move(A, C);

Hanoi(n-1, B, A, C);

}

}

void move(char x, char y)

/\*定义move函数\*/

{

printf("%c→%c ", x, y);

}

int main(void)

{

int n;

char A = 'A', B = 'B', C = 'C';

printf("Please input the amount of the dishes.\n");

scanf("%d", &n);

Hanoi(n, A, B, C);

return 0;

}

## 7 数组作为函数参数

### 7.1 释义

数组可以作为函数的参数使用，进行数据传送。数组用作函数参数有两种形式。

一种是把数组元素(下标变量)作为实参使用；

另一种是把数组作为函数的形参和实参使用。

数组元素就是下标变量，它与普通变量并无区别。因此它作为函数实参使用与普通变量时完全相同的，在发生函数调用时，把作为实参的数组元素的值传送给形参，实现单向的值传送。

示例：

判别一个整数数组a[10]={1, 2, 3, 4, -1, -2, -3, -4, 2, 3}中各元素的值，若大于0则输出该值，若小于等于0则输出0值。

实现：

#include <stdio.h>

int main (void)

{

void test(int v);

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

int i;

for (i = 0; i < 10; i++)

{

test (a[i]);

}

printf("\n");

return 0;

}

void test(int v)

{

if (v > 0)

{

printf("%d ", v);

}

}

### 7.2 用数组名作函数参数与用数组元素作实参的不同之处

(1)用数组元素作实参时，只要数组类型和函数的形参变量的类型一致，那么作为下标变量的数组元素的类型也和函数形参变量的类型是一致的。因此，并不要求函数的形参也是下标变量。换句话说，对数组元素的处理是按普通变量对待的。

然而，用数组名作函数参数时，则要求形参和相对应的实参都必须是类型相同的数组，都必须有明确的数组说明。当形参和实参二者不一致时，即会发生错误。

(2)在普通变量或下标变量作函数参数时，形参变量和实参变量是由编译系统分配的两个不同的内存单元。在函数调用时发生的值传送是把实参变量的值赋予形参变量。

在用数组名作函数参数时，不是进行值的传送(而是进行址传送)，即不是把实参数组的每一个元素的值都赋予形参数组的各个元素。

为什么？

因为实际上形参数组并不存在，编译系统不为形参数组分配内存。

那么，数据的传送是如何实现的呢？

在此之前我们曾介绍过，数组名就是数组的首地址。因此在数组名作函数参数时所进行的传送只是地址的传送，也就是说把实参数组的首地址赋予形参数组名。

形参数组名取得该首地址之后，也就等于有了实在的数组。实际上是形参数组和实参数组为统一数组，共同拥有一段内存空间。

示例：

#include <stdio.h>

void test (int b[10]);

void main()

{

int a[] = {2, 4, 6, 8, 10, 12, 14, 16, 18, 20};

test(a);

putchar('\n');

}

void test (int b[10])

{

int i = 0;

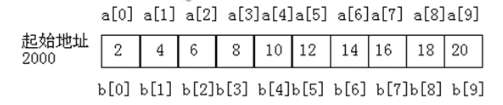
for (; i < 10; i++)

{

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

}

}



思考并讨论以下例题：

有一个一维数组score，内放10个学生成绩，求平均成绩(写一个average函数求平均成绩)。

实现：

#include <stdio.h>

double average(int b[10]);

int sum(int a[10]);

void main()

{

int score[10] = {2, 4, 6, 8, 10, 12, 14, 16, 18, 20};

double result;

result = average(score);/\*注意这里的score后面不能加上[10],否则会报错。这里要传递的是数组的首地址。\*/

printf("The average is %lf.\n", result);

}

int sum(int a[10]) /\*这里的a[10]可以写成a[],因为编译器只要知道定义的是个数组类型就可以。\*/

{

int i, sum = 0;

for (i = 0; i < 10; i++)

{

sum += a[i];

}

return sum;

}

double average(int b[10])

{

double result;

result = sum(b) / 10.0;/\*同样，这里的sum后面不能加[10]，否则也会报错，原因同上。\*/

return result;

}

## 8 局部变量和全局变量

### 8.1 局部变量

在一个函数内部定义的变量时内部变量，它只在本函数范围内有效，也就是说只有在本函数内才能使用它们，在此函数外是不能使用这些变量的。这称为：“局部变量”。

示例：

float f1( int a )/\*函数f1\*/

{

int b, c;/\*a, b, c有效\*/

……

}

char f2( int x, int y)/\*函数f2\*/

{

int i, j;/\*x, y, i, j有效\*/

}

void main()/\*主函数\*/

{

int m, n;/\*m, n有效\*/

}

说明：

(1)主函数中定义的变量(m, n)也只在函数中有效，而不因为在主函数中定义而在整个文件或程序中有效。主函数也不能使用其它函数中定义的变量。

(2)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。例如，上面在f1函数中定义了变量b和c，倘若在f2函数中也定义变量b和c，它们在内存中占不同的单元，互不混淆。

(3)形式参数也是局部变量。例如上面f1函数中的形参a，也只是在f1函数中有效。其他函数可以调用f1函数，但不可以引用f1函数的形参a。

(4)在一个函数内部，可以在复合语句(例如for()里面又有一个for())中定义变量，这些变量只在复合语句中有效，这种复合语句也称为“分程序”或“程序块”。

示例：

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\*/

…

}

…

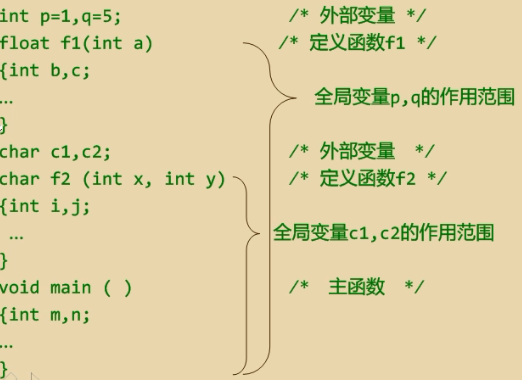
}

### 8.2 全局变量

在函数内定义的变量是局部变量，而在函数之外定义的变量称为外部变量，外部变量是全局变量(也称为全程变量)。

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

示例：



题目1：输入正方体的长宽高l,w,h。求体积及三个面,,的面积。

实现：

#include <stdio.h>

int s1, s2, s3;

int vs(int a, int b, int c)

{

int v;

v = a\*b\*c;

s1 = a\*b;

s2 = a\*c;

s3 = b\*c;

return v;

}

void main()

{

int v, l, w, h;

printf("\nintput length, width and height.\n");

scanf("%d %d %d", &l, &w, &h);

v = vs(l, w, h);

printf("\nv = %d, s1 = %d, s2 = %d, s3 = %d\n", v, s1, s2, s3);

}

还可以用如下程序实现：但要注意一些小问题：

#include <stdio.h>

double l, w, h;

void area(double x, double y, double z)

{

printf("面积分别为：%lf，%lf，%lf\n", l\*w, l\*h, w\*h);

}

double volume(double x, double y, double z)

{

double volume;

volume = x \* y \* z;

return (volume);

}

int main(void)

{

#if(0)

double volume, area;/\*若加上此句，程序会报错，错误报告：error C2063: 'area' : not a function error C2063: 'volume' : not a function

原因是系统会混淆volume和area的含义，认为后面的volume和area的含义不是函数，而是定义的变量。因此会报错。\*/

#endif

double a, b, c;

printf("请输入长宽高。\n");

scanf("%lf %lf %lf", &l, &w, &h);

area(l, w, h);

printf("体积为：%lf\n",volume( l, w, h));

return 0;

}

题目2：

有一个一维数组，内放10个学生成绩，写一个函数，求出平均分、最高分和最低分。

实现：

#include <stdio.h>

int score[10], max, min, i;

double average( int score[10] )

{

double result;

int sum = score[0];

i = 10;

while (--i)

{

sum += score[i];

}

result = sum / 10.0;

return (result);

}

void maxminscore( int score[10])

{

max = (score[0]>score[1])?score[0]:score[1];

min = (score[0]>score[1])?score[1]:score[0];

for (i = 2; i < 10; i++)

{

max = (max>score[i])?max:score[i];

min = (min>score[i])?score[i]:min;

}

}

void main()

{

printf("请输入这十个同学的分数，用空格隔开。\n");

for(i = 0; i < 10; i++)

{

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

}

maxminscore(score);

printf("平均分为：%lf\n最高分%d\n最低分%d\n", average(score), max, min);

}

参考答案：

#include <stdio.h>

float Max = 0, Min = 0;/\*定义全局变量\*/

void main()

{

float average(float array[], int n);

float ave, score[10];

int i;

for(i = 0; i < 10; i++)

{

scanf("%f", &score);

}

ave = average(score, 10);

printf("max = %6.2f\nmin = %6.2f\naverage = %6.2f\n", Max, Min, ave);

}

float average(float array[], int n)/\*定义函数，形参为数组\*/

{

int i;

float aver, sum = array[0];/\*如果这里的aver改为ave会不会和main函数中的ave重复呢？不会，原因是当该函数调用后，这个ave所占的内存会被释放掉。\*/

Max = Min = array[0];

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

{

if (array[i] > Max)

{

Max = array[i];

}

else if

{

Min = array[i];

}

sun += array[i];

}

aver = sum / n;

}

### 8.3 全局变量缺点的分析

建议不在必要时不要使用全局变量，原因如下：

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

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

它使得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及其值一起移过去。

但若该外部变量与其他文件的变量同名时，就会出现问题，降低了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”的渠道与外界发生关系外，没有其他渠道。

## 9 变量的存储类别

### 9.1 动态存储方式与静态存储方式

前面已经介绍了从变量的作用域(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)的、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开始时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需要进行动态的分配存储空间的方式。

用户存储空间可以分为三部分：

1.程序区

2.静态存储区

3.动态存储区

### 9.2 变量的存储类别

在C语言中每一个变量和函数有两个属性：数据类型和数据的存储类别。

对数据型(如整型、字符型等)。存储类别指的是数据在内存中存储的方式。

存储方式分为两大类：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。

#### 9.2.1 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(栈)，数据存储在动态存储区中。

函数中的形参和在函数中定义的变量(包括在复合语句中定义的变量)，都属此类，在调用该函数是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作为存储类别的声明。

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\*/

……

}

关键字auto可以省略，auto不写则隐含定义为“自动存储类别”，属于动态存储方式。

#### 9.2.2 用static声明局部变量

##### 9.2.2.1 声明方法

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。

这时就应该指定局部变量为“静态局部变量”，用关键字static进行声明。

示例：

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*该小程序考察静态局部变量的值\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <stdio.h>

int f(int a)

{

auto b = 0;/\*相当于auto int b = 0;也就是相当于int b = 0;局部变量存放在栈。\*/

static c = 3;/\*相当于auto int c = 3;也就是相当于int c = 3;静态变量存放在存储区。\*/

b = b + 1;/\*b == 1\*/

c = c + 1;/\*c == 4\*/

return (a+b+c);/\*2 + 1 + 4 == 7\*/

}

void main()

{

int a = 2, i;/\*这里的i也是局部变量所以它不会进入另一个函数中去。然而a却可以进入，原因是a作为实参传入上面的函数中了。\*/

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

{

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

}

}

##### 9.2.2.2 对静态局部变量的说明

(1)静态局部变量属于静态存储类别，在静态存储区内分配存储单元。在程序整个运行期间都不释放。

而自动变量(即动态局部变量)属于动态存储类别，占动态存储空间而不占静态存储空间，函数调用后即释放。

(2)对静态局部变量是在编译时赋初值的，即只复制一次，在程序运行时它已有初值。

以后每次调用函数时不再重新赋初值而只是保留上次函数调用结束时的值。

而对自动变量赋初值，不是在编译时进行的，而是在函数调用时进行，没调用一次函数重新给一次初值，相当于执行一次赋初值语句。

(其实静态变量常用于保存一些固定存储的值。)

(3)如在定义局部变量是不赋初值的话，则对静态局部变量来说，编译时自动赋初值0(对数值型变量)或空字符(对字符型变量)。

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。

这是由于每次函数调用结束后存储单元已释放，下次调用时又重新另分配存储单元，而所分配的单元中的值是不确定的。

(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。

示例：

输出1到5的阶乘值。

实现：

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* 打印1到5的阶乘值 \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <stdio.h>

int fac(int n)

{

static int f = 1;/\*f会保留上一个值\*/

f = f \* n;

return (f);

}

void main()

{

int i;

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

{

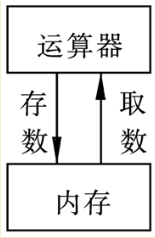
printf("%d! = %d\n", i, fac(i));

}

}

#### 9.2.3 register变量

一般情况下，变量(包括静态存储方式和动态存储方式)的值是存放在内存中的。



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

如果(这里说的是如果)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某局部变量)，则为存储变量的值要花费不少时间。

为了提高执行效率，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中取出参加运算，不必再到内存中去存取。由于对寄存器的存取速度远高于对内存的存取速度，因此这样做可以提高执行效率。这种变量叫做寄存器变量，用关键字register作声明。^\_^

示例：

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* 打印1到5的阶乘值 \*/

/\* 改进版本!! \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <stdio.h>

int fac(int n)/\*注意，这个版本不用static特性，所以把阶乘的过程放在fac()函数中实现。\*/

{

register int i, f = 1;

/\*使用register会提高速度，但是也不要用的太多，因为CPU中的寄存器很少，

当使用的变量定义到了里面，不够用的时候，剩余的就会变成普通的auto类别。\*/

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

{

f \*= i;

}

return (f);

}

void main()

{

int i;

for (i = 1; i <= 15; i++)

{

printf("%d! = %d\n", i, fac(i));

}

}

#### 9.2.4 extern声明外部变量

外部变量即是全局变量，它的作用域是从变量的定义处开始，到本程序文件结尾。

在此作用域内，全局变量可以为程序中各个函数所引用。编译时将外部变量分配在静态存储区。

有时需要用extern来声明外部变量，以拓展外部变量的作用域。

示例：

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* 用extern声明外部变量，拓展程序文件中的作用域。 \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <stdio.h>

int max(int x, int y)

{

int z;

z = x>y ? x : y;

return (z);

}

void main()

{

extern A, B;/\*试试去掉extern关键字\*/

printf("%d\n", max(A, B));

}

int A = 13, B = -8

进阶：在多文件的程序中声明外部变量

第一次将一个程序拆成多个文件！

示例：

第一个文件：

#include <stdio.h>

int A; /\*定义外部变量\*/

void main()

{

int power (int); /\*函数声明\*/

int b = 3, c, d, m;

printf("enter the number A and its power m:\n");

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

c = A \* b;

printf("%d \* %d = %d\n", A, b, c);

d = power(m);

printf("%d ^ %d = %d\n", A, m, d);

}

第二个文件：

extern A; /\*声明A为一个已定义的外部变量\*/

int power(int n)

{

int i, y = 1;

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

{

y \*= A;

}

return y;

}

#### 9.2.5 用static声明外部变量

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

若在上面的程序小改一下，把第一个文件的int A;改为static int A;这样，这里我们增加了static不让别的文件引用。再次运行的话，编译器会报错。错误报告：error LNK2001: unresolved external symbol \_A这样，第二个文件不能引用第一个文件的A变量，所以，编译器无法正常运行。

### 9.3 变量的声明和定义

对变量而言，声明与定义的关系稍微复杂一些。在声明部分出现的变量有两种情况：一种是需要建立存储空间的(如：int a;),另一种是不需要建立存储空间的(如：extern a;它的意思是告诉编译器：已经有一个a变量了，但是不在本文件中，而在另一个文件中。)。

前者称为“定义性声明”(defining declaration)，或简称定义(definition)。

后者称为“引用性声明”(referencing declaration)。

广义地说，声明包括定义，但并非所有的声明都是定义。

对“int a;”而言，它既是声明，又是定义。而对“extern a;”而言，它是声明而不是定义。(声明的意思是：对编译器说存在这么一个变量，而定义的含义是：对编译器说明存在这么一个变量而且给这个变量一定的值。)

一般为了叙述方便，把建立存储空间的声明称为定义，而把不需要建立存储空间的声明称为声明。显然指的声明是狭义的，即非定义性声明。

例如：

void main()

{

extern A;/\*是声明不是定义，声明A是一个已经定义的外部变量\*/

……

}

int A;/\*既是声明又是定义\*/

### 9.4 存储类别小结

不同角度作归纳：

(1)从作用域角度

局部变量 全局变量

局部变量：1.自动变量，即动态局部变量(离开函数值就消失)

2.静态局部变量(离开函数，值仍保留)

3.寄存器变量(离开函数，值就消失)可以加快速度

(形式参数可以定义为自动变量或寄存器变量)

全局变量：1.静态外部变量(只限本文件引用)

2.外部变量

(即非静态的外部变量，允许其他文件引用)

(2)从变量存在的时间(生存期)来分

动态存储空间：在调用函数时临时分配单元

静态存储空间：在调用程序整个运行时间都存在

动态存储：自动变量 (本函数内有效)

寄存器变量(本函数内有效)

形式参数 (本函数内有效)

静态存储： 静态局部变量(函数内有效)

静态外部变量(本文件内有效)

外部变量 (其他文件可引用)

(3)从变量值存放的位置来区分

内存中的静态存储区：静态局部变量

静态外部变量(函数外部静态变量)

外部变量(可为其他文件引用)

内存中的动态存储区：自动变量和形式参数

CPU中的寄存器：寄存器变量

(4)关于作用域和生存期的概念

作用域是从空间的角度，生存期是从时间的角度。

(5)static对局部变量和全局变量的作用域不同。

对局部变量来说，它使变量由动态存储方式改变为静态存储方式。而对全局变量来说，它使变量局部化(局部于本文件)，但仍为静态存储方式。

从作用域的角度看，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)。

## 10 内部函数和外部函数

函数本质上是全局的，因为一个函数要被另一个函数调用，但是，也可以指定函数不能被其他文件调用。

我们根据函数能否被其他源文件调用，将函数区分为内部函数和外部函数。

### 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。

即：static 类型标识符 函数名(形参表)

如：static int fun (int a, int b)

### 10.2 外部函数

(1)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。如函数首部可以写为extern int fun(int a, int b)这样，函数fun就可以为其他文件调用。C语言规定，如果在定义函数时省略extern，则隐含为外部函数。前面所用的函数都是外部函数。

(2)在需要调用此函数的文件中，用extern对函数作声明，表示该函数是在其他文件中定义的外部函数

示例：

有一个字符串，内有若干个字符，要求输入一个字符，程序将字符串中的该字符删去。

要求用外部函数实现！

实现：

第一个文件：

#include <stdio.h>

void main()

{

extern void enter\_string(char str[]);

extern void delete\_string(char str[], char ch);

extern void print\_string(char str[]);

/\*以上3行声明在本函数中将要调用的其他文件中定义的3个函数\*/

char c;

char str[80];

enter\_string(str);

scanf("%c", &c);

delete\_string(str, c);

print\_string(str);

}

第二个文件：

#include <stdio.h>

/\*定义外部变量enter\_string()\*/

void enter\_string(char str[80])

{

gets(str);/\*向字符数组输入字符串\*/

}

第三个文件：

/\*定义外部函数delete\_string()\*/

void delete\_string(char str[], char ch)

{

int i, j;

for(i = j = 0; str[i] != '\0'; i++)

{

if (str[i] != ch)

{

str[j++] = str[i];/\*效率会比较低，因为要实现字符串的拷贝。\*/

}

}

str[j] = '\0';

}

第四个文件：

#include <stdio.h>

void print\_string(char str[])

{

printf("%s\n", str);

}

关键步骤是第三个文件！

# 第七章 指针

## 1 概念和理解

### 1.1 概念

指针式C语言中的一个重要的概念，也是C语言的一个重要特色。

正确而灵活地使用它，可以有效地表示复杂的数据结构；能动态分配内存；能方便地使用字符串；有效而方便地使用数组……

掌握指针的应用，可以使程序简洁、紧凑、高效。可以说，不掌握指针就是没有掌握C的精华。

内存区的每一个字节有一个编号，这就是“地址”。如果在程序中定义了一个变量，在对程序进行编译时，系统就会给这个变量分配内存单元。

在C语言中，对变量的访问有两种方式，直接访问和间接访问。

直接访问如：a=5；

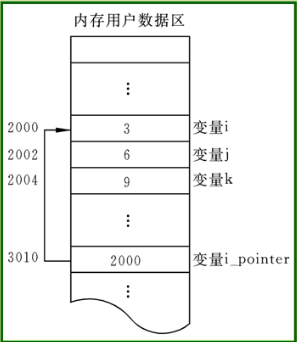
系统在编译时，已经对变量分配了地址，例如，若变量a分配的地址是2000，则该语句的作用就是把常数5保存到地址为2000的单元。

间接访问如：scanf(“%d”,&a);

调用函数时，把变量a的地址传递给函数scanf，函数首先把该地址保存到一个单元中，然后把从键盘接收的数据通过所存储的地址保存到a变量中。

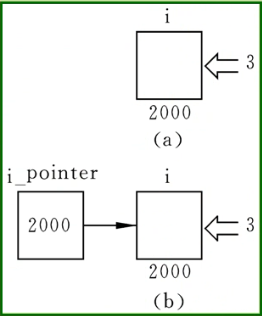
### 1.2 初识指针

在C语言中，指针是一种特殊的变量，它是存放地址的。假设我们定义了一个指针变量int \*i\_pointer用来存放整型变量i的地址。可以通过语句：i\_pointer = &i;



将i的地址(2000)存放到i\_pointer中。这时，i\_pointer的值就是2000，即变量i所占用单元的起始地址。

要存取变量i的值，可以采用间接方式：先找到存放“i的地址”的变量i\_pointer，从中取出i的地址2000，然后取出i的值3。



两个操作符“\*”和“&”

\*：取值操作符

&：取址操作符

如：

int i = 2000;

int \*pointer;

pointer = &i;

printf(“%d\n”,\*pointer);

### 1.3 指针与指针变量

知道了一个变量的地址，就可以通过这个地址来访问这个变量，因此，又把变量的地址称为该变量的“指针”。

C语言中可以定义一类特殊的变量，这些变量专门用来存放变量的地址，称为指针变量。

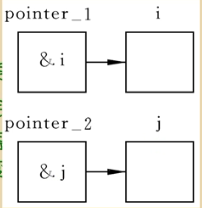
注意：指针变量的值(即指针变量中存放的值)是地址(即指针)。请区分“指针”和“指针变量”这两个概念。

下面都是合法的定义：

float \*pointer\_3;//pointer\_3是指向float型变量的指针变量

char \*pointer\_4;//pointer\_4是指向字符型变量的指针变量

可以用赋值语句使一个指针变量得到另一个变量的地址，从而使它指向一个该变量。如：



### 1.4 在定义指针变量时要注意两点

1.指针变量前面“\*”，表示该变量的类型为指针型变量。其一般形式为：

类型说明符 \*变量名；

其中，\*表示这是一个指针变量，变量名即为定义的指针变量名，类型说明符表示本指针变量所指向的变量的数据类型。

例如：float \*pointer\_1;指针变量名是pointer\_1,而不是\*pointer\_1。

2.在定义指针变量时必须指定基类型。

需要特别注意的是，只有整型变量的地址才能放到指向整型变量的指针变量中。下面的赋值是错误的：

float a;

int \*pointer\_1;

pointer\_1 = &a;/\*float 型变量的地址放到指向整型变量的指针变量中，错误\*/

### 1.5 指针变量的引用“&”

请牢记，指针变量中只能存放地址(指针)，不要将一个整数(或任何其他非地址类型的数据)赋给一个指针变量，否则编译器也会把该值当成一个地址来处理。

C语言中提供了地址运算符&来表示变量的地址。

其一般形式：&变量名；

如&a表示变量a的地址，&b表示变量b的地址。当然，变量本身必须预先声明。

示例：

#include <stdio.h>

void main()

{

int a, b;

int \*pointer\_1, \*pointer\_2;

a = 100;

b = 10;

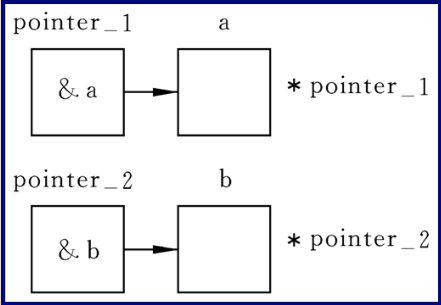
pointer\_1 = &a;

pointer\_2 = &b;

printf("%d, %d\n", a, b);

printf("%d, %d\n", \*pointer\_1, \*pointer\_2);

}



## 2 详细说明&和\*运算符

### 2.1 释义

如果已经执行了语句pointer\_1 = &a;

(1)&\*pointer\_1的含义是什么？

“&”和“\*”两个运算符的优先级别相同，但按自右而左方向结合，因此先进行\*pointer\_1的运算，它就是a，再执行&运算。

因此，&\*pointer\_1与&a相同，即变量a的地址。

如果有：

pointer\_2 = &\*pointer\_1;

它的作用是将&a(a 的地址)赋给pointer\_2，如果pointer\_2原来指向b，经过重新赋值后它已不再指向b了，而指向了a。

(2)\*&a的含义是什么？

先进行&a运算，得a的地址，在进行\*运算。即&a所指向的变量，也就是变量a。

\*&a和\*pointer\_1的作用是一样的，它们都等价于变量a。即\*&a与a等价。

(3)(\*pointer\_1)++相当于a++。

注意括号是必要的，如果没有括号，就成为了\*pointer\_1++,而++和\*为同一优先级别，而结合方向为自右至左，因此它相当于\*(pointer\_1++)。

由于++在pointer\_1的右侧，是“后加”，因此先对pointer\_1的原值进行\*运算，得到a的值，然后使pointer\_1的值改变，这样pointer\_1不再指向a了。

题目：输入a和b两个整数，按先大后小的顺序输出a和b。

实现：

#include <stdio.h>

void main()

{

int \*p1, \*p2, \*p, a, b;

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

p1 = &a;

p2 = &b;

if (a < b)

{

p = p1;

p1 = p2;

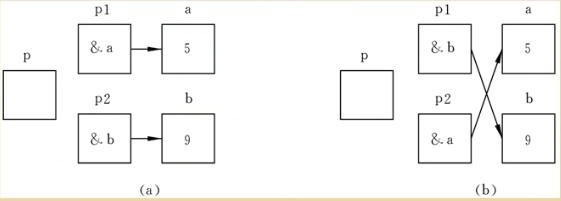
p2 = p;

}/\*此后的p1指向b,p2指向a\*/

printf("a = %d, b = %d\n", a, b);

printf("max = %d, min = %d\n", \*p1, \*p2);

}



题目：对输入的两个整数按大小顺序输出！这次用函数实现交换功能！

实现：

#include <stdio.h>

int main()

{

void compare(int a, int b);

int a, b;

printf("Please enter two integer numbers:\n");

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

compare(a, b);

return 0;

}

void compare(int a, int b)

{

int \* a\_pointer, \* b\_pointer, \* temp;

a\_pointer = &a;

b\_pointer = &b;

if(a < b)

{

temp = a\_pointer;

a\_pointer = b\_pointer;

b\_pointer = temp;

}

printf("%d %d\n", \* a\_pointer, \* b\_pointer);

}

题目：输入a、b、c3个整数，按大小顺序输出。

程序思考：假定a>b>c,那么做几次对比？

可以这样实现：

#include <stdio.h>

void compare(int \* a, int \* b);

void swap(int \* p1, int \* p2);

int main()

{

int a, b, c;

int \* a\_pointer, \* b\_pointer, \* c\_pointer;

a\_pointer = &a;

b\_pointer = &b;

c\_pointer = &c;

printf("Please enter three integer numbers:\n");

scanf("%d %d %d", a\_pointer, b\_pointer, c\_pointer);

compare(a\_pointer, b\_pointer);

compare(a\_pointer, c\_pointer);

compare(b\_pointer, c\_pointer);

printf("%d %d %d \n", \* a\_pointer, \* b\_pointer, \* c\_pointer);

return 0;

}

void compare(int \* a, int \* b)

{

if(\*a < \*b)

{

swap(a, b);

}

}

void swap(int \*pt1, int \*pt2)

{

int temp;

temp = \*pt1;

\*pt1 = \*pt2;

\*pt2 = temp;

}

同样也可以这样实现：(下面的程序是标准答案)

#include <stdio.h>

void main()

{

void exchange(int \*q1, int \*q2, int \*q3);

int a, b, c, \*p1, \*p2, \*p3;

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

p1 = &a;

p2 = &b;

p3 = &c;

exchange(p1, p2, p3);

printf("%d %d %d\n", a, b, c);

}

void exchange(int \*q1, int \*q2, int \*q3)

{

void swap(int \*pt1, int \*pt2);

if (\*q1 < \*q2)

{

swap(q1, q2);

}

if (\*q1 < \*q3)

{

swap(q1, q3);

}

if (\*q2 < \*q3)

{

swap(q2, q3);

}

}

void swap(int \*pt1, int \*pt2)

{

int temp;

temp = \*pt1;

\*pt1 = \*pt2;

\*pt2 = temp;

}

在这个程序中药注意：若要交换地址，则swap函数的定义方式如下：

void swap(int \* p1, int \* p2)

{

int \* temp;

temp = p1;

p1 = p2;

p2 = temp;

}

若要交换地址是行不通的，而要交换数值，所以swap函数采用原来程序的定义方式。(动态指针变量不能跨函数传递)

(上述文字有待修改)

## 3 数组与指针

### 3.1 基本含义

#### 3.1.1 基本使用方法

一个变量有地址，一个数组包含若干个元素，每个数组元素都在内存中占用存储单元，它们都有相应的地址。

指针变量既然可以指向变量，当然也可以指向数组元素(把某一元素的地址放到一个指针变量中 )。

所谓的数组元素的指针就是数组元素的地址。

定义一个指向数组元素的指针变量的方法，与以前介绍的指向变量的指针变量相同。

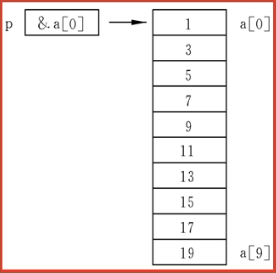
例如：

int a[10];(定义a为包含10个整型数据的数组)

int \*p;(定义p为指向整型变量的指针变量)

应该注意：如果数组为int型，则指针变量的基类型亦应该为int型。

p = &a[0];把a[0]元素的地址赋给指针变量p。也就是说使p指向a数组的第0号元素，见图：



引用一个数组元素，可以用：

(1)下标法，如a[i]形式；

(2)指针法，如\*(a + i)或\*(p + i)，表示指向第i个元素。

其中的a是数组名，p是指向数组元素的指针变量，其初值p = a。

**注意：数组名即“翻译成数组的第一个元素的地址！”**

示例：

假设有一个a数组，整型，有10个元素。要输出各元素的值有三种方法：

(1)下标法

(2)通过数组名计算数组元素地址，找出元素的值。

(3)用指针变量指向数组元素。

程序：

(1) 下标法

#include <stdio.h>

int main()

{

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

int i;

for(i = 0; i < 10; i++)

{

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

}

printf("\n");

}

(2) 通过数组名计算数组元素地址，找出元素的值。

#include <stdio.h>

int main()

{

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

int i;

for(i = 0; i < 10; i++)

{

printf("%d ", \*(a + i));

}

printf("\n");

}

(3) 用指针变量指向数组元素。

#include <stdio.h>

int main()

{

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

int i;

int \*p;

for(p = a; p < (a + 10); p++)

{

printf("%d ", \*p);

}

printf("\n");

}

思考题：

神马情况？

#include <stdio.h>

void main()

{

int \*p, i, a[10];

p = a;

for ( i=0; i < 10; i++ )

{

scanf("%d", p++);

}

printf("\n");

for ( i=0; i < 10; i++, p++)

{

printf("%d", \*p);

}

}

答案：输入之后没有将p的值变回a，可以在第12行加上p = a;进行修改。

在上面介绍过可以用数组名作函数的参数。

如：

void f(int arr[], int n)

{

…………

}

void main()

{

int array[10];

…………

f(array, 10);

}

f(int arr[], int n)

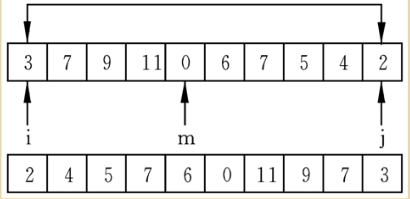
但在编译时是将arr按指针变量处理的，相当于将函数f的首部写成f(int \*arr, int n) (在VC编译器中默认将数组按指针变量来处理的)

以上两种写法是等价的。

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

示例：

将数组a中n个整数按相反顺序存放



程序实现：

#include <stdio.h>

void reverse(int a[], int n);

int main()

{

int a[] = {3, 7, 9, 11, 0, 6, 7, 5, 4, 2, 5};

int i;

printf("The original array:\n");

for (i = 0; i < 11; i++)

{

printf("%d ", \*(a+i));

}

printf("\n");

reverse(a, 11);

printf("The array has been inverted as follow:\n");

for (i = 0; i < 11; i++)

{

printf("%d ", \*(a+i));

}

printf("\n");

return 0;

}

void reverse(int a[], int n)/\*这里的int a[]也可以改为int \*a，原因是：函数只要获取数组的首地址就可以。即使int a[],这样其实也是传递一个数组的首地址。这时上面声明的时候也要改成这个形式。\*/

{

int i, j;

int temp;

for ( i = 0, j = n-1; i < j; i++, j--)

{

temp = \*(a+i);

\*(a+i) = \*(a+j);

\*(a+j) = temp;

}

}

练习：

从10个数中找出其中最大值和最小值。

版本一：采用下标法

#include <stdio.h>

int max(int a[], int n);

int min(int a[], int n);

int main()

{

int i, a[10];

printf("Please input ten nunbers:\n");

for(i = 0; i < 10; i++)

{

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

}

printf("The max is %d.\nThe min is %d.\n", max(a, 10), min(a, 10));

return 0;

}

int max(int a[], int n)

{

int i, max = a[n-1];

while(--n)

{

max = max>a[n-1]?max:a[n-1];

}

return max;

}

int min(int a[], int n)

{

int i, min = a[n-1];

while(--n)

{

min = min<a[n-1]?min:a[n-1];

}

return min;

}

版本二：用指针做参数

#### 3.1.2 小结

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

(1)形参和实参都用数组名，如：

void main()

{

int a[10];

f(a, 10);

}

void f(int x[], int n)

{

……

}

(2)实参用数组名，形参用指针变量。如：

void main()

{

int a[10];

f(a, 10);

}

f(int \*a, int n)

{

……

}

(3)实参形参都用指针变量。例如：

void main()

{

int a[10], \*p = a;

f(p, 10);

}

void f(int \*x, int n)

{

……

}

(4)实参为指针变量，形参为数组名。如：

void main()

{

int a[10], \*p = a;

…………

f(p, 10);

}

void f (int x[], int n)

{

…………

}

示例：

对数组中10个整数按由大到小的顺序排序。

实现：

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*对数组中10个整数按由大到小的顺序排序\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <stdio.h>

void paixu(int \*a, int n);

void exchange(int \*a, int \*b);

int main()

{

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

int i;

paixu(a, 10);

for(i = 0; i < 10; i++)

{

printf("%d ", \*(a+i));

}

printf("\n");

return 0;

}

void paixu(int \*a, int n)

{

int temp, i, j;

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

{

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

{

if ( \*(a+i) < \*(a+j) )

{

exchange(a+i, a+j);

}

}

}

}

void exchange(int \*a, int \*b)

{

int temp;

temp = \*a;

\*a = \*b;

\*b = temp;

}

标准答案：

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*对数组中10个整数按由大到小的顺序排序\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <stdio.h>

void sort(int x[], int n);

void main()

{

int \*p, i, a[10] = {3, 7, 9, 11, 0, 6, 7, 5, 4, 2};

printf("The original array:\n");

for (i = 0; i < 10; i++)

{

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

}

printf("\n");

p = a;

sort( p, 10 );

printf("The result is : \n");

for(p = a, i = 0; i < 10; i++)

{

printf("%d ", \*p);

p++;

}

printf("\n");

}

void sort(int x[], int n)

{

int i, j, k, t;

for(i = 0; i < n-1; i++)

{

k = i;

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

{

if (x[j] > x[k])

{

t = x[j];

x[j] = x[k];

x[k] = t;

}

}

}

}

### 3.2 多维数组和指针

用指针变量可以指向一维数组中的元素，也可以指向多维数组中的元素。

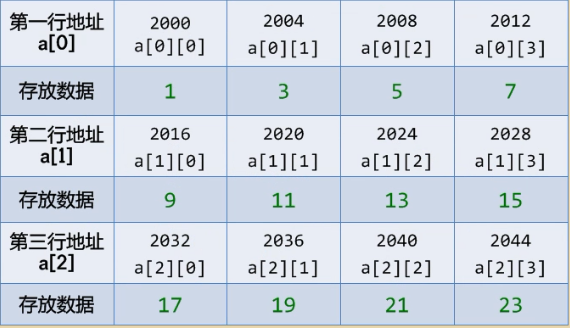
但在概念上和使用上，多维数组的指针比一维数组的指针要复杂一些。

先回顾一下多维数组的性质，可以认为二维数组是“数组的数组”，例：

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

则二维数组a是由3个一维数组所组成的。设二维数组的首行的首地址为2000

则有……





示例：

输出二维数组有关的值

实现：

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*输出二维数组的有关的值，牢记！！\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <stdio.h>

void main()

{

int a[3][4] = {{0, 1, 2, 3}, {4, 5, 6, 7}, {8, 9, 10, 11}};

/\*0行0列地址\*/

printf("a: %d\n", a);

printf("\*a: %d\n", \*a);

printf("a[0]: %d\n", a[0]);

printf("&a[0]: %d\n", &a[0]);

printf("&a[0][0]: %d\n", &a[0][0]);

/\*1行首地址\*/

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

printf("&a[1]: %d\n", &a[1]);

/\*1行0列元素a[1][0]首地址\*/

printf("\*(a+1): %d\n", \*(a+1));

printf("&a[1][0]: %d\n", &a[1][0]);

printf("a[1]: %d\n", a[1]);

/\*2行首地址\*/

printf("a+2: %d\n", a+2);

printf("&a[2]: %d\n", &a[2]);

/\*2行0列元素a[2][0]首地址\*/

printf("&a[2][0]: %d\n", &a[2][0]);

printf("a[2]: %d\n", a[2]);

printf("\*(a+2): %d\n", \*(a+2));

/\*1行1列元素a[1][1]首地址\*/

printf("a[1]+1: %d\n", a[1]+1);

printf("\*(a+1)+1: %d\n", \*(a+1)+1);

/\*1行1列元素a[1][1]的值\*/

printf("\*(a[1]+1): %d\n", \*(a[1]+1));

printf("\*(\*(a+1)+1): %d\n", \*(\*(a+1)+1));

}

/\*注意：不同的系统运行此程序会有不同的结果。原因是定义的地址会不同\*/

### 3.3 指向多维数组元素的指针变量

把二维数组a分解为一维数组a[0],a[1],a[2]之后，设p为指向二维数组的指针变量。可定义为：

int (\*p)[4]

它表示p是一个指针变量，它指向包含4个元素的一维数组。若指向第一个一维数组a[0]，其值等于a，a[0]，或&a[0][0]等。而p+i则指向一维数组a[i]。

从前面的分析可以得出\*(p+i)+j是二维数组i行j列的元素的地址，而\*(\*(p+i)+j)则是i行j列元素的值。

二维数组指针变量说明的一般形式为：

类型说明符 (\*指针变量名)[长度]

其中“类型说明符”为所指数组的数据类型。“\*”表示其后的变量是指针类型。“长度”表示二维数组分解为多个一维数组时，一维数组的长度，也就是二维数组的列数。

示例：

用指针变量输出二维数组元素的值

实现：

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*输出二维数组所有的值，牢记!!\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <stdio.h>

int main()

{

int a[3][4] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11};

int (\*p)[4];/\*尝试考虑int \*p[4]的意思\*/

int i, j;

p = a;

for (i = 0; i < 3; i++)/\*行循环\*/

{

for(j = 0; j < 4; j++)/\*列循环\*/

{

printf("%2d ", \*(\*(p+i)+j));

}

printf("\n");

}

printf("\n");

return 0;

}

练习：

通过输入指定行数和列数，打印出二维数组对应任一行任一列元素的值。

实现：

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* 通过输入指定行数和 \*/

/\* 列数，打印出二维数组对 \*/

/\* 应任一行任一列元素的值。\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <stdio.h>

int main()

{

int a[3][4] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11};

int (\*p)[4];

int m = 0;

int i = -1, j = -1;

p = a;

while( (i<0 || i>2) || (1 != m) )

{

printf("i = ");

m = scanf("%d", &i);

getchar();

}

while( (j<0 || j>3) || (1 != m))

{

printf("j = ");

m = scanf("%d", &j);

getchar();

}

printf("a[%d][%d] = %d\n", i, j, \*(\*(p+i)+j));

return 0;

}

## 4 字符串与指针

(1)用字符数组存放一个字符串，然后输出该字符串。

例题：

定义一个字符数组，对它初始化，然后输出该字符串。

实现：

#include <stdio.h>

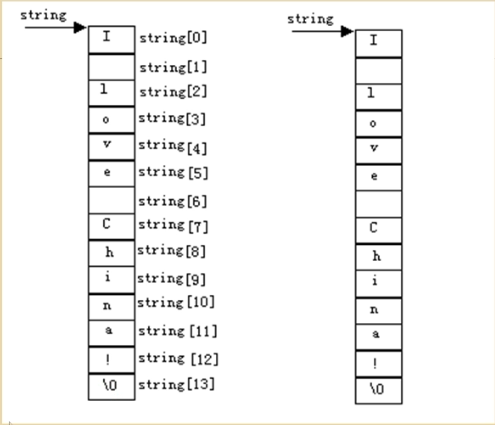
void main()

{

char string[] = "I love China!";

printf("%s\n", string);

}



(2)用字符指针指向一个字符串

例题：

可以不定义字符数组，而定义一个字符指针。用字符指针指向字符串中的字符。

实现：

#include <stdio.h>

void main()

{

char \*string = "I love China!";/\*注意：不能定义成char \*string[] = "I love China!";\*/

printf("%s\n", string);/\*注意：这里不要写成printf("%s\n", \*string);\*/

}

第(2)种方法要优于第(1)种方法

对字符串中的字符的存取，可以用下标方法，也可以用指针方法！

下标法举例：将字符串a复制为字符串b

实现：

#include <stdio.h>

void main()

{

char a[] = "I love China!", b[40];

int i;

for (i=0; \*(a+i) != '\0'; i++)

{

\*(b+i) = \*(a+i);

}

\*(b+i) = '\0';/\*表示字符串结束，这一句是非常必要的。\*/

printf("String a is:%s\n", a);

printf("String b is:");

for (i=0; b[i] != '\0'; i++)

{

printf("%c", b[i]);

}

printf("\n");

}

指针方法举例：将字符串a复制为字符串b

实现：

#include <stdio.h>

void main()

{

char a[] = "I love China!", b[40], \*p1, \*p2;

int i;

p1 = a;

p2 = b;

for (; \*p1 != '\0'; p1++, p2++)

{

\*p2 = \*p1;

}

\*p2 = '\0';

printf("String a is:%s\n", a);

printf("String b is:");

for (i=0; b[i] != '\0'; i++)

{

printf("%c", b[i]);

}

printf("\n");

}

例题：

用函数调用实现字符串的复制

(1)用字符数组作参数

实现：

#include <stdio.h>

void main()

{

void copy\_string(char from[], char to[]);

char a[] = "I am a teacher.";

char b[] = "You are a student.";

printf("String a = %s\nString b = %s\n", a, b);

printf("Copy string a to string b:\n");

copy\_string (a, b);

printf("\nString a = %s\nString b = %s\n", a, b);

}

void copy\_string(char from[], char to[])

{

int i = 0;

while(from[i] != '\0')

{

to[i] = from[i];

i++;

}

to[i] = '\0';

}

(2)形参用字符指针变量

实现：(注意：运行本段程序会出现问题！！！)

#include <stdio.h>

void main()

{

void copy\_string(char \*from, char \*to);

char \*a = "I am a teacher.";

char \*b = "You are a student.";/\*这里的char \*b若改成char b[],便不会出错。这是为什么？\*/

printf("String a = %s\nString b = %s\n", a, b);

printf("Copy string a to string b:\n");

copy\_string (a, b);

printf("\nString a = %s\nString b = %s\n", a, b);

}

void copy\_string(char \*from, char \*to)

{

for(; \*from != '\0'; from++, to++)

{

\*to = \*from;

}

\*to = '\0';

}

## 5 关于a[]和\*a的一些区别

重要！！！！！

五大内存分区：

在C和C++中，内存分为5个区，它们分别是堆、栈、自由存储区、全局/静态存储区和常量存储区。

栈，就是那些由编译器在需要的时候分配，在不需要的时候自动清除的变量的存储区。里面的变量通常是局部变量、函数参数等。

堆，就是那些由new分配的内存块，它们的释放编译器不去管，由我们的应用程序去控制，一般一个new就要对应一个delete。如果程序员没有释放掉，那么在程序结束后，操作系统会自动回收。

自由存储区，就是那些由malloc等分配的内存块，它和堆是十分相似的，不管它是用free来结束自己的生命的。

全局/局部存储区，全局变量和静态变量被分配到同一个内存中，在以前的C语言中，全局变量又分为初始化的和未初始化的，在C++里面没有这个区分了，它们共同占用同一块内存区。

常量存储区，这是一块比较特殊的存储区，它们里面存放的是常量，不允许修改(当然，你要通过非正当的手段也可以修改，而且方法很多)。

若声明的是字符串，在VC++编译器中会自动分配到常量存储区中，而常量存储区中的常量是不容改变的。所以是不可写(written)的。

到fishC网站上阅读这篇文章。<http://home.fishc.com/space.php?uld=9&do=blog&ld=182>

优化：对copy\_string 函数还可以作简化

方法一：

void copy\_string(char \*from, char \*to)

{

while( (\*to = \*from) != '\0' )

{

to++;

from++;

}

/\*先将from里面的值赋值给to然后和'\0'比较，

如果不相等，则执行循环体(即指针指向下一个元素)，

直到相等的时候退出循环。\*/

}

方法二：

void copy\_string(char \*from, char \*to)

{

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

{

;

}

/\*要看懂这句话，首先要知道\*和++的优先级。

显然，\*的优先级要比++高，所以，才会

实现和上面的程序一样的结果。\*/

}

方法三：

void copy\_string(char \*from, char \*to)

{

while( \*from != '\0' )

{

\*to++ = \*from++;

}

\*to = '\0';

}

方法四：

void copy\_string(char \*from, char \*to)

{

while( \*to++ = \*from++ )

/\*这里没有使复制字符结束的语句(如\*to = '\0';)

但仍然可以实现字符的复制过程， 原因是结束标志'\0'

相当于整型数字0，当复制进行到'\0'字符时，while后面

的括号内的值为0，跳出循环。这是个非常不错的方法。\*/

{

;

}

}

方法五：

void copy\_string(char \*from, char \*to)

{

for( ; \*to++ = \*from++; )

/\*这里没有使复制字符结束的语句(如\*to = '\0';)

但仍然可以实现字符的复制过程， 原因是结束标志'\0'

相当于整型数字0，当复制进行到'\0'字符时，for后面

的括号内的第二个语句值为0，跳出循环。这是个非常不错的方法。\*/

{

;

}

}

方法六：

void copy\_string(char from[], char to[])

{

char \*p1, \*p2;

p1 = from;

p2 = to;

while( (\*p2++ = \*p1) != '\0' )

{

;

}

}

可以通过查看它们在VC++中的反汇编过程比较它们的执行效率(即CPU执行的动作越少，效率越高。)，综合比较它们谁更好一些。

## 6 对使用字符指针变量和字符数组的讨论

虽然用字符数组和字符指针变量都能实现字符串的存储和运算，但它们二者之间是有区别的，不应该混为一谈……

主要概括起来有以下几点：

1.字符数组有若干个元素组成，每个元素中方一个字符，而字符指针变量中存放的是地址(字符串第1个字符的地址)，绝不是将字符串放到字符指针变量中。

2.赋值方式。对字符数组只能对各个元素赋值，不能用以下办法对字符数组赋值。

char str[20];

str= “I love China!”;

而对字符指针变量，可以采用下面方法赋值：

char \* a;

a = “I love China!”;

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.对字符指针变量赋初值：

char \* a = “I love China!”;

等价于

char \* a;

a = “I love China!”;

而对数组的初始化：

char str[20];

str[]= “I love China!”;

4.如果定义了一个字符数组，在编译时为它分配内存单元，它有确定的地址。而定义一个字符变量时，给指针变量分配的内存单元，在其中可以存放一个字符变量的地址，也就是说，该指针变量可以指向一个字符型数据，但如果并未具体指向一个确定的字符数据。

如：

char str[10];

scanf(“%s”,str);

以上是完全可以的！

而常有人用下面的方法，目的是想输入一个字符串，虽然一般也能运行，但这种方法是危险的：

char \*a;

scanf(”%s”,a);

5.指针变量的值是可以改变的。

例如：

#include <stdio.h>

int main(void)

{

char \* a = "I love China!";

printf("%s\n", a);

a += 7;

printf("%s\n", a);

return 0;

}

另外需要说明的是，若定义了一个指针变量，并使它指向一个字符串，就可以用下标形式引用指针变量所指向的字符串中的字符。

示例：

#include <stdio.h>

int main(void)

{

char \* a = "I love China!";

int i;

printf("The sixth character is %c\n\n", a[5]);

for( i=0; a[i] != '\0'; i++ )

{

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

}

printf("\n");

return 0;

}

## 7 指向函数的指针

用函数指针变量调用函数

可以用指针变量指向整型变量、字符串、数组，也可以指向一个函数。一个函数在编译时被分配一个入口地址。这个函数的入口地址就称为函数的指针。

示例：

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*比较a和b的大小，求大值\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <stdio.h>

#if(0)

int main(void)

{

int max(int, int);

int a, b, c;

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

c = max(a, b);

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

return 0;

}

#endif

int max(int a, int b)

{

if (a > b)

{

return a;

}

else

{

return b;

}

}

/\*将上面的函数改成如下的函数\*/

#if(1)

int main(void)

{

int max(int, int);

int (\*p)();

int a, b, c;

p = max;

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

c = (\*p)(a, b);

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

return 0;

}

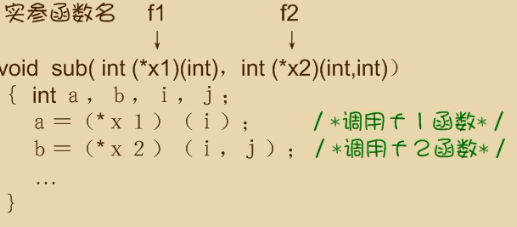
#endif

函数指针变量常用的用途之一是把指针作为参数传递到其他函数。

前面介绍过，函数的参数可以是变量、指向变量的指针变量、数组名、指向数组的指针变量等。

现在介绍指向函数的指针也是可以作为参数，以实现函数地址的传递，这样就能够在被调用的函数中使用实参函数。

举个例子：



它的原理可以简述如下：有一个函数(假设函数名为sub)，它有两个形参(x1和x2)，定义x1和x2为指向函数的指针变量。在调用函数sub时，实参为两个函数名f1和f2，给形参传递的是函数f1和f2的地址。这样在函数sub中就可以调用f1和f2函数了。

示例：

设一个函数process，在调用它的时候，每次实现不同的功能。(有点类似多态)

输入a和b两个数，第一次调用process时找出a和b中的大者，第二次找出其中小者，第三次求a与b之和。

实现：

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*设一个函数process，在调用它的时候，每次实现不同的功能。\*/

/\*输入a和b两个数，第一次调用process时找出a和b中的大者， \*/

/\*\*\*\*\*\*\*\* 第二次找出其中小者，第三次求a与b之和。 \*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <stdio.h>

int main(void)

{

int max(int, int);/\*函数声明\*/

int min(int, int);/\*函数声明\*/

int add(int, int);/\*函数声明\*/

void process( int, int, int(\*fun)() );/\*函数声明\*/

int a, b;

printf("Enter a and b;");

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

printf("max = ");

process(a, b, max);

printf("min = ");

process(a, b, min);

printf("sum = ");

process(a, b, add);

return 0;

}

int max(int x, int y)

{

int z;

if (x > y)

{

z = x;

}

else

{

z = y;

}

return z;

}

int min(int x, int y)

{

int z;

if (x > y)

{

z = y;

}

else

{

z = x;

}

return z;

}

int add(int x, int y)

{

int z;

z = x + y;

return z;

}

void process( int x, int y, int(\*fun)(int x, int y) )

{

int result;

result = fun(x, y);

printf("%d\n", result);

}

## 8 返回指针值的函数

一个函数可以带回一个整型值、字符型、实型值等，也可以带回指针型的数据，即地址。其概念也与以前类似，只是带回的值的类型是指针类型而已。

这种带回指针值的函数，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);

例题：

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。

实现：

#include <stdio.h>

int main(void)

{

double score[][4] = {{60.0, 70.0, 80.5, 90.5}, {56.0, 89.0, 67.0, 88.0}, {34.2, 78.5, 90.5, 76.2}};

double \*search(double(\*pointer)[4], int n);

double \*p;

int i, m;

printf("Please enter the number of student: ");

scanf("%d", &m);

printf("The scores of No.%d are :\n", m);

p = search(score, m);

for( i=0; i<4; i++)

{

printf("%5.2f\t", \*(p+i));

}

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

return 0;

}

double \*search(double (\*pointer)[4], int n)

{

double \*pt;

pt = \*(pointer+n);

return pt;

}

程序升级：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。

## 9 指针函数和函数指针的区别

这两个概念都是简称：

指针函数是指带指针的函数，即本质是一个函数。

函数指针式指向函数的指针变量，因此函数指针本身首先应是指针变量，只不过该指针变量指向函数。

## 10 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，也就是说，指针数组中的每一个元素都相当于一个指针变量。一维指针数组的定义形式为：

类型名 数组名[数组长度];

例如：

int \* name[4];

示例：

#include <stdio.h>

int main(void)

{

int a[5] = {1, 3, 5, 7, 9};

int \* name[5] = {&a[0], &a[1], &a[2], &a[3], &a[4]};

int i;

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

{

printf("%d ", \*name[i]);

}

printf("\n\n");

return 0;

}

例题：将下面字符串按字母顺序(由小到大)输出。

char \*name[] = {“Kangge”, “I love Kangge”, “Kangge wansui!”, “fuck you!”, “Thank you!”};

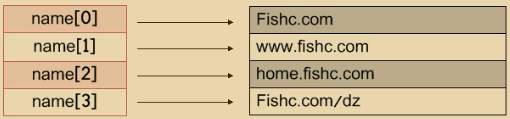
提示：

一，我们可以声明一个数组指针来指向；

二，将排序利用strcmp()函数来解决；

三，各个功能抽象为函数或文件。

我们可以声明指针数组：



实现：

#include <stdio.h>

#include <string.h>

int main(void)

{

void sort(char \*name[], int n);

void print(char \*name[], int n);

char \*name[] = {"Kangge", "I love Kangge", "Kangge wansui!", "fuck you!", "Thank you!"};

int n = 5;

sort(name, n);

print(name, n);

return 0;

}

void sort(char \*name[], int n)

{

char \*temp;

int i, j, k;

for (i=0; i<n-1; i++)

{

k = i;

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

{

if (strcmp(name[k], name[j]) > 0)

{

k = j;

}

if (k != i)

{

temp = name[i];

name[i] = name[k];

name[k] = temp;

}

}

}

}

void print(char \*name[], int n)

{

int i;

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

{

printf("%s\n", name[i]);

}

}

## 11 指向指针的指针

### 11.1 指向指针数据的指针的定义

怎样定义一个指向指针数据的指针变量呢？

形式可以如：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)，显然\*p是指针变量的定义形式。如果没有最前面的\*，那就是定义了一个指向字符数据的指针变量。

示例：

#include <stdio.h>

int main(void)

{

char \*name[] = {"Kangge", "I love Kangge", "Kangge wansui!", "fuck you!", "Thank you!"};

int \*\*p;

int i;

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

{

p = name + i;

printf("%s\n", \*p);

}

}

### 11.2 指针数组作main函数的形参

指针数组的一个重要应用是作为main函数的形参。在以往的程序中，main函数的第一行一般写成以下形式：

void main()

括弧中是空的。实际上， main函数可以有参数。

例如：void main(int argc, char \*argv[])

argc和argv就是main函数的形参。

main函数是由操作系统调用的。那么，main函数的形参的值从何处得到呢？

显然不可能在程序中得到。实际上实参是和命令一起给出的。也就是在一个命令行中包括命令名和需要传给main含糊的参数。

命令行的一般形式：

命令名 参数1 参数2 ……参数n

示例：

比较下面两个程序的不同

1.

#include <stdio.h>

int main(int argc, char \*argv[])

{

while (argc > 1)

{

++argv;

printf("%s\n", argv);

--argc;

}

}

2.

#include <stdio.h>

#include <stdlib.h>

void main(int argc, char \*argv[])

{

int i;

printf("The number of the string is: %d\n", argc-1);

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

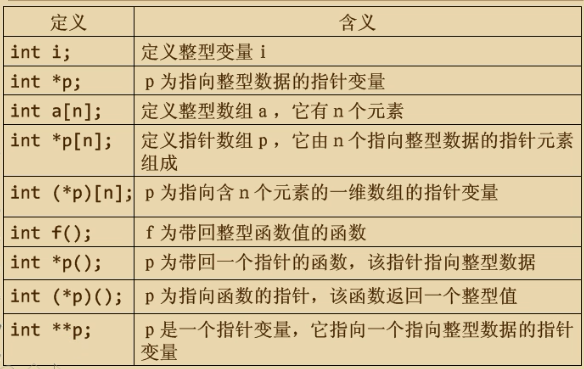
{

printf("The string %d is : %s\n", i, argv[i]);

}

}

### 11.3 有关指针的数据类型的小结



#### 11.3.1指针运算小结

一、指针变量加(减)一个整数

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

二、

指针变量赋值

将一个变量地址赋给一个指针变量。如：

p = &a;(将变量a的地址赋给p)

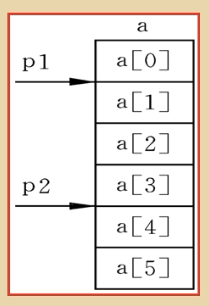
p = array;(将数组array首元素地址赋给p)

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)

三、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;



四、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。

五、两个指针变量比较

若两个指针变量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。

#### 11.3.2 关于void类型

##### 11.3.2.1 void的作用

void真正发挥的作用在于：

(1)对函数返回的限定；

(2)对函数参数的限定。

例如：void abc(void);

##### 11.3.2.2 void指针和const指针

ANSIC新标准增加了一种“void”指针类型，即不指定它是指向哪一种类型数据的指针变量。

例如：void \*p;

表示指针变量p不指向一个确定的类型数据，它的作用仅仅是用来存放一个地址。

void指针可以指向任何类型数据。也就是说，可以用任何类型的指针直接给void指针赋值。但是，如果需要将void指针的值赋给其他类型的指针，则需要进行强制类型转换。

##### 11.3.2.3 关于const的三个例子

例子一：

#include <stdio.h>

void main(void)

{

const char \* str = "Welcome to LiaoYang!\n\n";

/\*这个语句的含义是：声明一个名为str的指针变量，

它指向一个字符型常量，初始化str为指向字符串"

Welcome to LiaoYang!\n\n"\*/

printf("\n\n%s", str);

#if (0)

str[0] = 'w';/\*这条语句是错误的，但可以改变str指针的值\*/

#endif

str = "I love KangKang!\n\n";/\*合法!!\*/

printf("\n\n%s", str);

}

例子二：

#include <stdio.h>

void main(void)

{

char \* const str = "Welcome to LiaoYang!\n\n";

/\*常量指针式一个固定的指针，不可以改变它的值，但它所指向的数据可以改变。\*/

str[0] = 'w';/\*合法!!\*/

#if (0)

str = "I love KangKang!\n\n";/\*非法!!\*/

#endif

printf("\n\n%s", str);

}

/\*window 下禁止对常量进行重写的，本程序语法无错误，但是运行起来会有错误。\*/

例子三：

#include <stdio.h>

void main(void)

{

char \* const str = "Welcome to LiaoYang!\n\n";

/\*常量指针式一个固定的指针，不可以改变它的值，但它所指向的数据可以改变。\*/

str[0] = 'w';/\*非法!!\*/

str = "I love KangKang!\n\n";/\*非法!!\*/

printf("\n\n%s", str);

}

hmemcpy() 被称为“万能断点”，在win9x时代风靡一时。(搞破解时经常用到该函数)。在windowNT架构的时候，它就变成了memcpy(),但是现在很少会有程序用它进行账号的密码的拷贝等等，原因是它很容易被下断点，容易被拷贝源码。

# 第八章 预处理

在前面各章中，已多次使用过以“#”好开头的预处理命令。

如包含命令#include <stdio.h>,宏定义命令#define PI 3.1415926等。

在源程序中这些命令都放在函数之外，而且一般都放在源文件的前面，它们成为预处理部分。

## 1 无参宏定义

无参宏的宏名后不带参数。

其定义的一般形式为：

#define 标识符 字符串

其中的“#”表示这是一条预处理命令。

凡是以“#”开头的均为预处理命令。

define为宏定义命令。“标识符”为所定义的宏名。“字符串”可以是常数、表达式、格式串等。

例如：

#define PI 3.1415926

它的作用是指定标识符PI来代替数3.1415926.

在编写源程序时，所有的3.1415926都可由PI代替，而对源程序作编译时，将先由预处理程序进行宏代换，即用3.1415926表达式去置换所有的宏名PI，然后再进行编译。

示例：

#include <stdio.h>

#define PI 3.1415926

void main()

{

double s;

int r;

printf("Please enter the radius:");

scanf("%d", &r);

s = PI \* r \* r;

printf("%g\n\n", s);

}

对于宏定义还要说明一下几点：

1.宏定义是用宏名来表示一个字符串，在宏展开时又以该字符串取代宏名，这只是一种简单的代换，字符串中可以含任何字符，可以是常数，也可以是表达式，预处理程序对它不做任何检查。如有错误，只能在编译已被宏展开后的源程序时发现。

2.宏定义不是说明或语句，在行末不必加分号，如加上分号则连分号也一起置换。

3.宏定义必须卸载函数之外，其作用域为宏定义命令起到源程序结束。如要终止起作用域可使用#undef命令。

4.宏名在源程序中若用引号括起来，则预处理程序不对其作红代换。

示例：

#include <stdio.h>

#define PI 3.1415926

void fun(void);

void main()

{

double s;

int r;

printf("Please enter the radius:");

scanf("%d", &r);

s = PI \* r \* r;

printf("\n\nThe area of the roundness = %g\n\n", s);

fun();

}

/\*如果在这里加上#undef PI,则下面的代码中的PI就会没有被定义到。\*/

void fun(void)

{

printf("Now the PI = %g\n\n", PI);

printf("PI\n\n");/\*PI在引号中应该是表示常量字符串，不替换……\*/

}

5.宏定义允许嵌套，在宏定义的字符串中可以使用已经定义的宏名。在宏展开时由预处理程序层层代换。

6.习惯上宏名用大写字母表示，以便于与变量区别。但也允许用小写字母。

示例：

#include <stdio.h>

#define PI 3.1415926

#define S PI \* r \* r

void fun(void);

void main()

{

double s;

int r;

printf("Please enter the radius:");

scanf("%d", &r);

s = S;

printf("\n\nThe area of the roundness = %g\n\n", s);

fun();

}

void fun(void)

{

printf("Now the PI = %g\n\n", PI);

printf("PI\n\n");

}

/\*本程序就用到了嵌套\*/

7.可用宏定义表示数据类型，使书写更方便。

例如：

#define INTEGER int

注意：宏定义表示数据类型和用typedef定义数据说明符的区别

区别：宏定义只是简单的字符串代换，是在预处理完成的，而typedef是在编译时处理的，它不是做简单的代换，而是对类型说明符重新命名。被命名的标识符具有类型定义说明的功能。typedef是一个语句。

示例：

#include <stdio.h>

#define PIN1 char \*

typedef char \* PIN2;

void main()

{

PIN1 x, y;/\*相当于char \* x, y;\*/

PIN2 a, b;/\*相当于char \* a, char \* b;\*/

printf("By #define : %d %d\n\n", sizeof(x), sizeof(y));

printf("By typedef : %d %d\n\n", sizeof(a), sizeof(b));

}

8.对“输出格式”作宏定义，可以减少书写麻烦。

示例：

#include <stdio.h>

#define P printf

#define D "%d\n"

#define F "%f\n"

void main()

{

int a = 5, c = 8, e = 11;

float b = 3.8, d = 9.7, f = 21.08;

P(D F, a, b);

P(D F, c, d);

P(D F, e, f);

}

例题：

不用循环和递归，实现打印0到999。

#include <stdio.h>

#define A(x) x;x;x;x;x;x;x;x;x;x;

int main(void)

{

int n = 0;

A(A(A(printf("%d ", n++))));/\*每层A()中printf("%d ", n++);被执行了10次，共有三层嵌套，所以是10^3次。\*/

return 0;

}

## 2 带参数的宏定义

C语言允许宏带有参数。在宏定义中的参数称为形式参数，在宏调用中的参数称为实际参数。

对带参数的宏，在调用中，不仅要展开，而且要用实参去代换形参。

带参宏定义的一般形式为：

#define 宏名(形参表) 字符串

带参宏调用的一般形式为：

宏名(实参表);

例如：

#define M(y) y \* y + 3 \* y /\*宏定义\*/

……

k = M(5); /\*宏调用\*/

……

在宏调用时，用实参5去代替形参y，经预处理宏展开后的语句为：

k = 5 \* 5 + 3 \* 5;

示例：

#include <stdio.h>

#define MAX(a, b) (a > b) ? a : b

int main(void)

{

int x, y, max;

printf("input two numbers: ");

scanf("%d %d", &x, &y);

max = MAX(x, y);

printf("The max is %d\n\n", max);

return 0;

}

对于带参的宏定义有以下问题需要说明：

1.带残宏定义中，宏名和形参表之间不能有空格出现。

例如把：

#define MAX(a, b) (a > b) ? a : b

写为：

#define MAX (a, b) (a > b) ? a : b

试试看。

2.在带参宏定义中，形式参数不分配内存单元，因此不必做类型定义。而宏调用中的实参有具体的值。要用它们去代换形参，因此必须做类型说明。

这是与函数中的情况是不同的。在函数中，形参和实参是两个不同的量，各有自己的作用域，调用时要把实参值赋予形参，进行“值传递”。而在带参宏中，只是符号代换，不存在值传递的问题。

3.在宏定义中的形参是标识符，而宏调用中的实参可以是表达式。

示例：

#include <stdio.h>

#define SAY(y) (y)/\*^是异或的意思\*/

void main()

{

int i = 0;

char say[] = {73, 32, 108, 111, 118, 101, 32, 75, 97, 110, 103, 75, 97, 110, 103, 33, 33, 0};

while (say[i])

{

say[i] = SAY(say[i]\*1);/\*相当于say[i]) ^ say[i] ^ say[i].在括号内加上\*1想说的是括号内可以放表达式。\*/

i++;

}

printf("\n\t%s\n\n", say);

}

4.在宏定义中，**字符串内的形参通常要用括号括起来以避免出错。**在上例中的宏定义中(y)\*(y)表达式的y都用括号括起来，因此结果是正确的。如果去掉括号，把程序改为以下形式：

#include <stdio.h>

#define SQ(y) (y) \* (y) /\*y\*y试试\*/

void main()

{

int a, sq;

printf("Input a number :");

scanf("%d", &a);

sq = SQ(a + 1);/\*若上面的#define SQ(y) (y) \* (y)改成 #define SQ(y) y \* y 则结果会发生改变。原因是这里的就会变成sq = a + 1 \* a + 1;\*/

printf("sq = %d\n", sq);

}

/\*由于以上的原因，一般宏定义时要把参数扩上括号。\*/

将上面例子中的题目进行变态：我们现在要160÷SQ(y)

#include <stdio.h>

#define SQ(y) (y) \* (y)

void main()

{

int a, sq;

printf("Input a number :");

scanf("%d", &a);

sq = 160 / SQ(a + 1);/\*这样替换就等价于：160 / (a + 1) \* (a + 1);从左向右运算，则只没有变。为了避免这样的结果，一般在宏定义时这样实现：#define SQ(y) ((y) \* (y))\*/

printf("sq = %d\n", sq);

}

由于以上原因，我们建议不经常使用宏定义，但是为了避免出错，应尽量的多加括号。

5.带参的宏和带参函数很相似，但有本质上的不同，除上面已谈到的个点外，把同一表达式用函数处理与用宏处理两者的结果有可能是不同的。

比较下面的两个程序的不同，并说出原因。

(1)

#include <stdio.h>

void main()

{

int i = 1;

int SQ(int y);

while (i <= 5)

{

printf("%d\n", SQ(i++));

}

}

int SQ(int y)

{

return ((y) \* (y));

}

(2)

#include <stdio.h>

#define SQ(y) ((y) \* (y))

void main()

{

int i = 1;

while (i <= 5)

{

printf("%d\n", SQ(i++));

}

}

6.宏定义也可以定义多个语句，在宏调用时，把这些语句又代换到源程序内。

示例：

#include <stdio.h>

#include <string.h>

#define STR(s1, s2, s3, sum) strcat(strcat(strcat(sum, s1), s2), s3);/\*strcat();函数把第二个字符串粘贴到第一个字符串的后面。\*/

void main()

{

char str1[] = "I ", str2[] = "love ", str3[] = "KangKang!!", str[40] = "";

STR(str1, str2, str3, str);/\*strcat(strcat(strcat(sum, str1), str2), str3);\*/

printf("\n\tstr1 = %s\n\tstr2 = %s\n\tstr3 = %s\n\tstr = %s\n\n\n", str1, str2, str3, str);

}

## 3 文件包含

尽管我们很熟悉，但对文件包含命令还要说明一下几点：

1.一个include命令只能指定一个被包含文件，若有多个文件要包含，则需要多个include命令。

2.文件包含允许嵌套，即在一个被包含的文件中又可以包含另一个文件。

3.包含命令中的文件名可以用双引号括起来，也可以用尖括号括起来。例如：以下写法都是允许的：

#include “stdio.h”

#include <math.h>

但是这两种形式是有区别的：使用尖括号表示在包含文件目录中去查找(包含目录是由用户在设置环境时设置的)，而不在源文件目录中去查找；

使用双引号则表示首先在当前的源文件目录中查找，若未找到才到包含目录中去查找。用户编程时可根据自己文件所在的目录来选择某一种命令形式。

## 4 条件编译

预处理程序提供了条件编译的功能。可以按不同的条件去编译不同的程序部分，因而产生不同的目标代码文件。这对于程序的移植和调试是很有用的。

条件编译有三种潜规则，下面分别介绍：

第一种形式：

#ifdef 标识符

程序段1

#else

程序段2

#endif

它的功能是，如果标识符已被#define命令定义过则对程序段1进行编译；否则对程序段2进行编译。

如果没有程序段2(它为空)，本格式中的#else可以没有，即可以写为：

#ifdef 标识符

程序段

#endif

第二种形式：

#ifndef 标识符

程序段1

#else

程序段2

#endif

示例：

#include <stdio.h>

#define CORRECT "Kangge"/\*如果该语句没有写，也不会造成程序错误，原因是后面有#ifndef CORRECT#define CORRECT "Kangge"#endif。这主要是针对大型程序，如果忘记前面的定义，可以在后面#ifndef……\*/

void main()

{

char str[40];

int cmp(char \* str1, char \* str2);

printf("Please enter the name you like the best :");

scanf("%s", str);

#ifndef CORRECT

#define CORRECT "Kangge"

#endif

if (cmp(str, CORRECT) == 0)

{

printf("\nYeah! You are a smart man!!\n\n");

}

else

{

printf("\nYou fool! Man!!\n\n");

}

}

int cmp(char \* str1, char \* str2)

{

int i = 0, j = 0;

while (str1[i])

{

while (str2[j] == str1[i])

{

i++;

j++;

if (!str2[j])

{

return 0;

}

}

j = 0;

i++;

}

return -1;

}

第三种形式：

#if 常量表达式

程序段1

#else

程序段2

#endif

示例：

#include <stdio.h>

#define ROUND 1/\*如果ROUND后面是1则输出的是圆面积，如果后面是0则输出的是正方形的面积。\*/

#define PI 3.1415926

void main()

{

int r;

double s;

printf("input a number:");

scanf("%d", &r);

#if ROUND

s = r \* r \* PI;

printf("Area of round is :%6.5f\n\n", s);

#else

s = r \* r;

printf("Area of square is:%6.5f\n\n", s);

#endif

}

## 5 本章小结

1.预处理功能C语言特有的功能，它是在对源程序正是编译前由预处理程序完成的。程序员在程序中用预处理命令来调用这些功能。

2.宏定义是用一个标识符来表示一个字符串，这个字符串可以是常量、变量或表达式。在宏调用中将用该字符串代换宏名。

3.宏定义可以带有参数，宏调用时是以实参代换形参。而不是“值传送”。(函数是“值传送”)

4.为了避免宏代换时发生错误，宏定义中的字符串加括号，字符串中出现的形式参数两边也应加括号。

5.文件包含使预处理的一个重要功能，它可以用来把多个源文件接成一个源文件进行编译，结果将生成一个目标文件。

6.条件编译允许只编译源程序中满足条件的程序段，使生成的目标程序较短，从而减少了内存的开销并提高了程序的效率。

7.使用预处理功能便于程序的修改、阅读、移植和调试，也便于实现模块化程序设计。

# 第九章 结构体与共用体

## 1 概述

问题定义：

有时需要将不同的数据组合成一个有机的整体，以便于应用。如：

一个学生有 学号/姓名/年龄/地址等属性int num;char name[20];char sex;int age;int char addr[30];



定义一个结构的一般形式：

struct 结构名

{

成员表列

};

成员表列由若干个成员组成，每个成员都是该结构的一个组成部分。对每个成员也必须做类型说明，其形式为：

类型说明符 成员名;

示例：

struct student

{

int num;

char name[20];

char sex;

int age;

float age;

char addr[30];

}

可以采取以下3种方法定义结构体类型变量：

(1)先声明结构体类型再定义变量名

例如：

定义了student1和student2为struct student 类型的变量，即它们具有struct student 类型的结构。

示例：

struct student

{

int num;

char name[20];

char sex;

int age;

float score;

char addr[30];

}student1, student2;

在定义结构体变量后，系统会为之分配内存单元。例如：

student1和student2在内存中个占？个字节。

(4 + 20 + 1 + 4 + 4 + 30 = 67)。(用sizeof测量一下，结果不是67，而是68，这是编译器的优化结果以及对齐。)

(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;

示例：

struct student

{

int num;

char name[20];

char sex;

int age;

float age;

char addr[30];

}student1, student2;

(3)直接定义结构体类型变量

其一般形式为：

struct

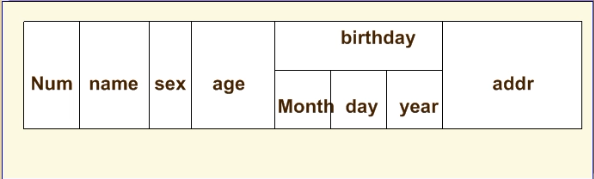
{

成员表列

}变量名表列;

即不出现结构体名。

看图定义



这是一个嵌套的定义

首先定义一个结构date，由month(月)、day(日)、year(年)三个成员组成。

在定义并说明变量boy1和boy2时，其中的成员birthday被说明为date结构类型。成员名可与程序中其它变量同名，互不干扰。

struct date

{

int month;

int day;

int year;

};

struct

{

int num;

char name[20];

struct date birthday;

float score;

}boy1, boy2;

## 2 结构体变量的引用

在定义 了结构体变量以后，当然可以引用这个变量。但应遵守以下规则：

(1)不能将一个结构体变量作为一个整体进行输入和输出。

正确引用结构体变量中的成员的方式为：

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 = 100;

“.”是成员(分量)运算符，它在所有的运算中优先级最高，因此可以把student1.num作为一个整体来看待。上面赋值语句的作用是将整数100赋给student1变量中的成员num。

(2)如果成员本身有属于一个结构体类型，则要用若干个成员运算符，一级一级地找到最低的一级的成员。只能对最低级的成员进行赋值或存取以及运算。

对上面定义的结构体变量student1，可以这样访问各成员：

student1.num

student1.birthday.month

(3)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;

(4)可以引用结构体变量成员的地址，也可以引用结构体变量的地址。

示例：

#include <stdio.h>

int main(void)

{

struct student

{

int num;

char \*name;

char sex;

float score;

} boy1;

boy1.num = 007;

boy1.name = "Jane";

printf("The address of struct is :%o \n", &boy1);

printf("The address of num is :%o \n", &boy1.num);

return 0;

}

但不能用以下语句整体读入结构体变量：

scanf(“%d, %s, %c, %d, %f, %s”, &student1);

结构体变量的地址主要用作函数参数，传递结构体变量的地址。

## 3 结构体变量的初始化

我们可以这么对结构体进行初始化！

示例：

#include <stdio.h>

int main(void)

{

struct student

{

int num;

char \* name;

char sex;

float score;

} boy1, boy2 = {102, "Jane", 'M', 98.5};

boy1 = boy2;

printf("Number = %d\nName = %s\nScore = %f\n", boy1.num, boy1.name, boy1.score);

printf("\n\n");

printf("Number = %d\nName = %s\nScore = %f\n", boy2.num, boy2.name, boy2.score);

return 0;

}

## 4 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。

结构体数组与以前介绍的数值型数组不同之处在于每个数组元素都是一个结构体类型的数据，它们都分别包括各个成员(分量)项。

示例：

#include <stdio.h>

#include <stdlib.h>

#define NUM 3

struct person

{

char name[20];

char phone[10];

};

int main(void)

{

struct person man[NUM];

int i;

for (i=0; i<NUM; i++)

{

printf("input name:\n");

gets(man[i].name);

printf("input phone:\n");

gets(man[i].phone);

}

printf("\tname\t\t\t\tphone\n\n");

for (i=0; i<NUM; ++i)

{

printf("%20s\t\t\t%20s\n", man[i].name, man[i].phone);

}

return 0;

}

## 5 定义结构体数组

和定义结构体变量的方法相仿，只需说明其为数组即可。例如：

struct student

{

int num;

char name[20];

char sex;

int age;

float score;

char addr[30];

};

struct student student[3];

也可以这样：

struct student

{

int num;

char name[20];

char sex;

int age;

float score;

char addr[30];

} student[3];

## 6 结构体数组的初始化

与其他类型的数组一样，对结构体数组可以初始化。例如：

struct student

{

int num;

char name[20];

char sex;

int age;

float score;

char addr[30];

} stu[2] = {

{101, "LiLin", 'M', 18, 87.5, "Beijing"},

{102, "Zhang", 'F', 18, 99, "Shanghai"}

};

当然，数组的初始化也可以用以下形式：

struct student

{

int num;

…

}

struct student str[]{{…},{…},{…}};

即先声明结构体类型，然后定义数组为该结构体类型，在定义数组时初始化。

例题：

对候选人得票的统计程序。设有3个候选人，每次输入一个得票的候选人的名字，要求最后输出各人得票结果。

实现：

## 7 指向结构类型数据的指针

一个结构体变量的指针就是该结构体变量所占据的内存段的起始地址。

可以设一个指针变量，用来指向一个结构体变量，此时该指针变量的值是结构体变量的起始地址。(指针伟大哈)

指针变量也可以用来指向结构体数组中的元素。

结构体指针变量说明的一般形式：

struct 结构名 \* 结构指针变量名

例如，在前面的例题中定义了stu这个结构，如要说明一个指向stu的指针变量pstu，可以写为：

struct stu \*pstu;

当然也可以在定义stu结构时说明pstu。与前面讨论的各类指针变量相同，结构指针变量也必须要先赋值后才能使用。

赋值是把结构变量的首地址赋予该指针变量，不能把结构名赋予该指针变量。

如果boy是被说明为stu类型的结构变量，则：

pstu = &boy;

是正确的。

pstu = &stu;

是错误的。

结构名和结构变量是两个不同的概念，不能混淆。结构名只能表示一个结构形式，编译系统并不对它分配内存空间。

只有当某个变量被说明为这种类型的结构时，才对该变量分配存储空间。

因此上面&stu这种写法是错误的，不可能去取一个结构名的首地址。有了结构指针变量，就能方便地访问结构变量的各个成员。

其访问的一般形式为：

(\*结构指针变量).成员名

或为：

结构指针变量->成员名

例如：

(\*pstu).num

或者：

pstu->num

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。

#include <stdio.h>

struct stu

{

int num;

char \* name;

char sex;

float score;

} boy1 = {102, "kangge", 'M', 78.5};

int main(void)

{

struct stu \*pstu;

pstu = &boy1;

printf("Number = %d\nName = %s\n", boy1.num, boy1.name);

printf("Sex = %c\nScore = %f\n\n", boy1.sex, boy1.score);

printf("Number = %d\nName = %s\n", (\*pstu).num, (\*pstu).name);

printf("Sex = %c\nScore = %f\n\n", (\*pstu).sex, (\*pstu).score);

printf("Number = %d\nName = %s\n", pstu->num, pstu->name);

printf("Sex = %c\nScore = %f\n\n", pstu->sex, pstu->score);

return 0;

}

## 8 结构指针变量作函数参数

将一个结构体变量的值传递给另一个函数有3种方法：

(1)用结构体变量的成员作为参数

(2)用结构体变量作实参

(3)用指向结构体变量(或数组)的指针作实参，将结构体变量(或数组)的地址传递给形参

示例：

有一个结构体变量stu,内含学生号、姓名和3门课程的成绩。通过调用函数printf中将它们输出。

实现：

#include <stdio.h>

struct student

{

long int ID;

char name[20];

double score[3];

};

void print(struct student stu)

{

printf("%ld, %s, %2.1f, %2.1f, %2.1f\n\n", stu.ID, stu.name, stu.score[0], stu.score[1], stu.score[2]);

}

int main(void)

{

struct student stu = {1120010118, "徐振康", {79.0, 65.0, 88.0}};

print(stu);

return 0;

}

将上面的例子改用指向结构体变量的指针作实参。

#include <stdio.h>

struct student

{

long int ID;

char name[20];

double score[3];

};

void print(struct student \* p)

{

printf("%ld, %s, %2.1f, %2.1f, %2.1f\n\n", p->ID, p->name, p->score[0], p->score[1], p->score[2]);

}

int main(void)

{

struct student stu = {1120010118, "徐振康", {79.0, 65.0, 88.0}};

struct student \* p = &stu;

print(p);

return 0;

}

## 9 动态存储分配[重点]

在数组一章中，曾介绍过数组的长度是预先定义好的，在整个程序中固定不变。C语言中不允许动态数组类型。

例如：

int a[n];是错误的。

用变量表示长度，想对数组的大小作动态说明，这是错误的。

但是在实际编程中，往往会发生这种情况，即所需要的内存空间取决于输入的数据，而无法预先确定。

所以对于这种问题，用数组的办法是很难解决的！为了解决上述问题，C语言提供了一些内存管理函数，这些内存管理函数可以按需要动态的分配内存空间，也可把不再使用的空间回收待用，为有效地利用内存资源提供了手段。

常用的内存管理函数有以下三个：

1.分配内存空间函数malloc、calloc

2.释放内存空间函数free

### 9.1 malloc函数

函数原型为void \* malloc(unsigned int size);

其作用是在内存的动态存储区中分配一个长度为size的连续空间(size是一个无符号整数)

此函数的返回值是一个指向分配域起始地址的指针(类型为void)。

如果此函数未能成功地执行(例如内存空间不足)，则返回空指针(NULL)。

### 9.2 calloc函数

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。

用calloc函数可以为一维数组开辟动态存储空间，n为数组元素个数，每个元素长度为size。

### 9.3 free函数

函数原型为void free(void \* p);

其作用是释放由p指向的内存区，使这部分内存能被其他变量使用。

p是最近一次调用calloc或malloc函数时返回的值。

free函数无返回值。

### 9.4 链表

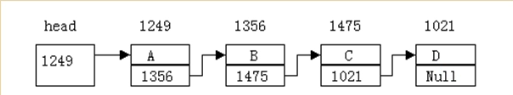
什么是链表？

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：

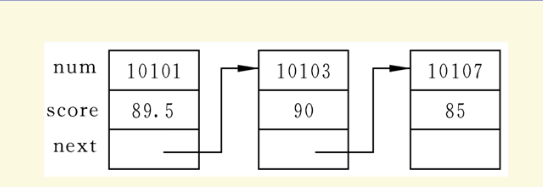
头指针：存放一个地址，该地址指向第一个元素

结点：用户重要的实际数据和链接节点的指针



示例：

我们尝试根据下图建立链表：



实现：

#include <stdio.h>

struct student

{

long num;

float score;

struct student \* next;

};

int main(void)

{

struct student a, b, c, \* head;

a.num = 10101;

a.score = 89.5;

b.num = 10103;

b.score = 90;

c.num = 10107;

c.score = 85;

head = &a;

a.next = &b;

b.next = &c;

c.next = NULL;

do

{

printf("%ld %5.1f\n", head->num, head->score);

head = head->next;

}while (head);

return 0;

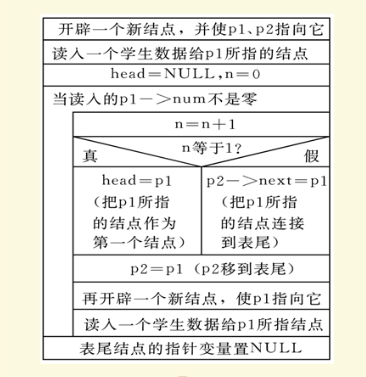
}

### 9.5 建立动态链表

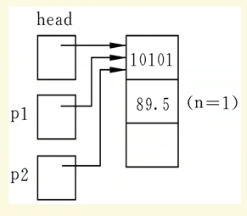
所谓建立动态链表是指在程序执行过程中从无到有地建立起一个链表，即一个一个地开辟结点和输入结点数据，并建立起前后相连的关系。

作业：

根据下面的分析写一个程序建立一个含有学生(学号，成绩)数据的单向动态链表。(约定：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。)



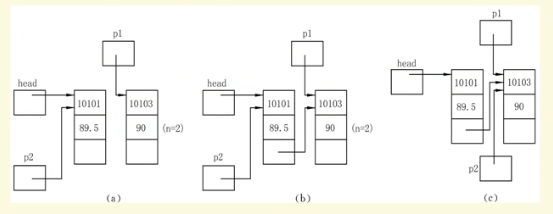
如果输入的p1->num不等于0，则输入的是第一个结点数据(n = 1)，令head = p1;即把p1的值赋给head，也就是使head也指向新开辟的结点p1所指向的新开辟的结点就成为链表中第一个结点。



在开辟另一个结点并使p1指向它，接着输入该结点的数据。

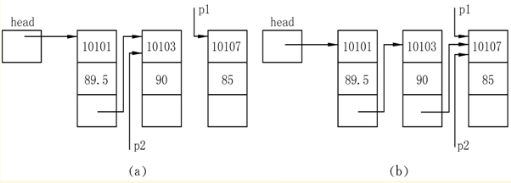
如果输入的p1->num0,则应链入第2个结点(n=2),将新结点的地址赋给第一个结点的next成员。

接着使p2 = p1，也就是使p2指向刚才建立的结点。



再开辟一个结点并使p1指向它，并输入该结点的数据。

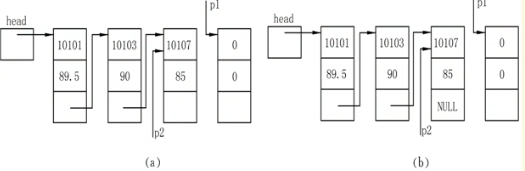
在第三次循环中，由于n=3(n1)，又将p1的值赋给p2->next,也就是将第三个结点连接到第二个结点之后，并使p2=p1，使p2指向最后一个结点。



再开辟一个新的结点，并使p1指向它，输入该结点的数据。由于p1->num的值为0，不再执行循环，此新结点不应被连接到链表中。

将NULL赋给p2->next。

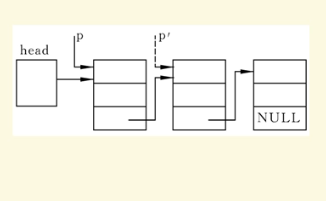
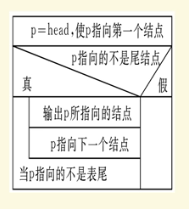
建立链表过程至此结束，p1最后所指向的结点未链入链表中，第三个结点的next成员的值为NULL,它不指向任何结点。



### 9.6 实现链表的输出

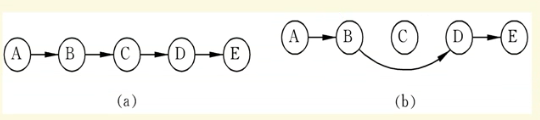
首先要知道链表第一个结点的地址，也就是要知道head的值。

然后设计一个指针变量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，在输出，直到链表的尾结点。



### 9.7 对链表的删除操作

从一个动态链表中删去一个结点，并不是真正从内存中把它抹掉，而是把它从链表中分离开来，只要撤销原来的链接关系即可。



题目：写一个函数以删除动态链表中指定的结点。

解题思路：

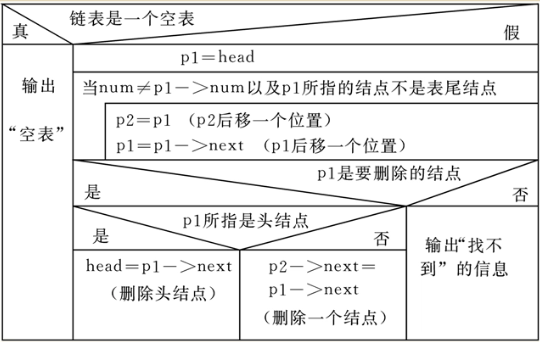
一、从p指向的第一个结点开始，检查该结点中的num值是否等于输入的要求删除的那个学号。

二、如果相等就将该结点删除，如果不相等，就将p后移一个结点，在如此进行下去，直到遇到表尾为止。

三、可以设两个指针变量p1和p2，先使p1指向第一个结点。

四、如果要删除的不是第一个结点，则使p1后移指向下一个结点(将p1->next赋给p1)，在此之前应将p1的值赋给p2，使p2指向刚才检查过的那个结点。

五、将以上几点我们综合得出算法流程图：



对链表结点的删除操作实现

实现：

#include <stdio.h>

#include <malloc.h>

#include <stdlib.h>

#define LEN sizeof(struct student)

struct student \* creat();//创建链表

struct student \* del(struct student \* head, int num);//del函数用于删除结点， \*head即链表的头指针，num是要删除的结点num。

void print(struct student \* head);//打印链表

struct student

{

int num;

float score;

struct student \* next;

};

int n;//全局变量，用来记录存放了多少数据。

int main(void)

{

struct student \*stu, \*p;

int n;

stu = creat();

p = stu;

print(p);

printf("Please enter the num to delete:");

scanf("%d", &n);

print(del(p, n));

printf("\n\n");

system("pause");

return 0;

}

struct student \* creat()

{

struct student \* head;

struct student \* p1, \* p2;

p1 = p2 = (struct student \*)malloc(LEN);//LEN是student结构的大小

printf("Please enter the num:");

scanf("%d", &p1->num);

printf("Please enter the score:");

scanf("%f", &p1->score);

head = NULL;

n = 0;

while (p1->num)

{

n++;

if (1 == n)

{

head = p1;

}

else

{

p2->next = p1;

}

p2 = p1;

p1 = (struct student \*)malloc(LEN);

printf("\nPlease enter the num:");

scanf("%d", &p1->num);

printf("Please enter the score:");

scanf("%f", &p1->score);

}

p2->next = NULL;

return head;

}

void print(struct student \* head)

{

struct student \*p;

printf("\nTHere are %d records!\n\n", n);

p = head;

if (head)

{

do

{

printf("学号为%d的成绩:%f\n\n", p->num, p->score);

p = p->next;

}while (p);

}

}

struct student \* del(struct student \* head, int num)

{

struct student \* p1, \* p2;

if (NULL == head)

{

printf("\nThis list is null!\n");

goto end;

}

p1 = head;

while (p1->num != num && p1->next != NULL)

{

p2 = p1;

p1 = p1->next;

}

if (num == p1->num)

{

if (p1 == head)

{

head = p1->next;

}

else

{

p2->next = p1->next;

}

printf("\nDelete No:%d succeed!\n", num);

n--;

}

else

{

printf("%d not been found!\n\n", num);

}

end:

return head;

}

### 9.8 对链表的插入操作

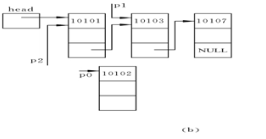
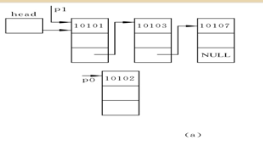
对链表的插入是指将一个结点插入到一个已有的链表中。

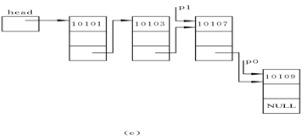
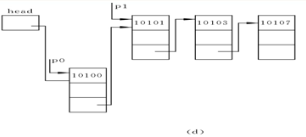
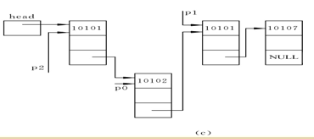
为了能够正确插入，必须解决两个问题：

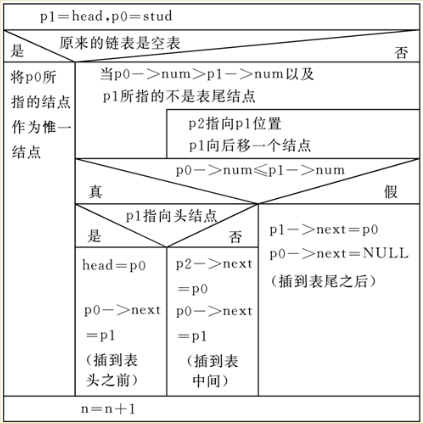
1.怎样找到插入的位置；

2.怎样实现插入。

我们可以先用指针变量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, 此时将p1后移， 并使p2指向刚才p1所指向的结点。







课后练习：

尝试实现插入程序并尝试制作一个学生管理系统。

## 10 共用体

### 10.1 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;

例如:

union data

{

int i;

char ch;

float f;

}a, b, c;

或

union data

{

int i;

char ch;

float f;

};

union data a, b, c;

### 10.2 共用体和结构体的比较

结构体变量所占内存长度是各个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/整型变量占4个字节)，而不是各占4+1+4=9个字节。

只有先定义了共用体变量才能引用它，而且不能引用公用体变量，而只能引用共用体变量中的成员。

例如：前面先定义了a, b, c为公用体变量

a.i(引用共用体变量中的整型变量i)

a.ch(引用共用体变量中的字符变量ch)

a.f(引用共用体变量中的实型变量f)

### 10.3 共用体类型数据的特点

同一个内存段可以用来存放几种不同类型的成员，但在每一瞬间只能存放其中一种，而不是同时存放几种。

共用体变量中起作用的成员是最后一次存放的成员，在存入一个新的成员名就失去作用。

共用体变量的地址和它的各成员的地址都是同一地址。

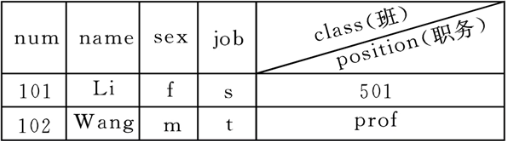
不能对共用体变量名赋值，也不能企图引用变量名来得到一个值，又不能在定义共用体变量时对它初始化。

不能把共用体变量作为函数参数，也不能使函数带回共用体变量，但可以使用指向共用体变量的指针

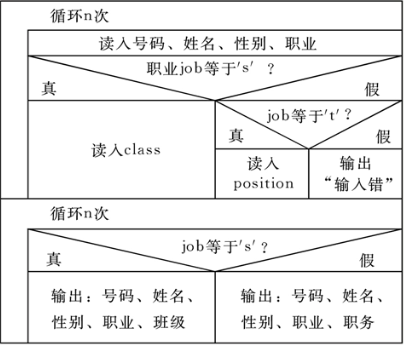
共用体类型可以出现在结构体类型定义中，也可以定义共用体数组。反之，结构体也可以出现在共用体类型定义中，数组也可以作为共用体的成员。

### 10.4 应用情况

设有若干个人员的数据，其中有学生和教师。学生的数据中包括：姓名、号码、性别、职业、班级。教师的数据包括：姓名、号码、性别、职业、职务。可以看出，学生和教师所包含的数据是不同的。现要求把它们放在同一表格中。



处理算法



## 11 枚举类型

在实际问题中， 有些变量的取值被限定在一个有限的范围内。

例如：一个星期内只有七天，一年只有十二个月，一个班级每周有六门课程等等。

如果把这些量说明为整型，字符型或其他类型显然是不妥当的。

为此，C语言提供了一种称为“枚举”的类型。

设有变量a, b, c被说明为上述的weekday，可采用下述人意重方式：

enum weekday{sun, mou, tue, wed, thu, fri, sat};

enum weekday a, b, c;

或者为：

enum weeday {sun, mou, tue, wed, thu,fri, sat}a, b, c;

或者为：

enum {sun, mou, tue, wed, thu, fri, sat}a, b, c;

在“枚举”类型的定义中列举出所有可能的取值，被说明为该“枚举”类型的变量取值不能超过定义的范围。

应该说明的是， 枚举类型是一种基本数据类型，而不是一种构造类型，因为它不能再分解为任何基本类型。

在枚举值表中应罗列出所有可用值。这些值也称为枚举元素。

示例：

/\*枚举类型\*/

#include <stdio.h>

int main(void)

{

enum weekday{sun, mon, tue, wed, thu, fri, sat} a, b, c;

a = sun;/\*a = 1这样是不行的……虽然编译不报错，但是它违背了本身的意义。\*/

b = mon;

c = tue;

printf("%d, %d, %d", a, b, c);

printf("\n\n");

return 0;

}

示例：

#include <stdio.h>

int main(void)

{

enum boy {Tom, Danny, Gan, LiLei} month[31], j;

int i;

j = Tom;

for (i=1; i<=30; i++)

{

month[i] = j;

j++;

if (j > LiLei)

{

j = Tom;

}

}

for (i=1; i<=30; i++)

{

switch (month[i])

{

case Tom: printf("%4d %s\t", i, "Tom");

break;

case Danny: printf("%4d %s\t", i, "Danny");

break;

case Gan: printf("%4d %s\t", i, "Gan");

break;

case LiLei: printf("%4d %s\t", i, "LiLei");

break;

default:

break;

}

}

printf("\n");

return 0;

}

## 12 typedef定义类型

### 12.1 用typedef声明新的类型名来代替已有的类型名

#### 12.1.1 声明INTEGER为整型

typedef int INTEGER

#### 12.1.2 声明结构体类型

typedef struct

{

int month;

int day;

int year;

}DATE;

#### 12.1.3 声明NUM为整型

typedef int NUM[100];

#### 12.1.4 声明STRING为字符指针类型

typedef char \* STRING;

#### 12.1.5 声明POINTER为指向函数的指针类型，该函数返回整型值。

typedef int (\*POINTER)();

关于typedef可以声明各种类型名，但不能用来定义变量。

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。

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，然后再需要用到它们的文件中用#include 命令把它们包含进来。

(题外话：函数名称和函数的地址是同一个东西。例如：

#include <stdio.h>

typedef void (\*p)();

void f(void);

int main(void)

{

p p1;

p1 = f;//等价于p1 = &f；

(p1)();

return 0;

}

void f(void)

{

printf("函数名称和函数地址是同一个东西"!);

}

)

### 12.2 用typedef定义类型的方法

先按定义变量的方法写出定义体(如：int i)

将变量名换成新类型名(例如：将i换成COUNT)。

在最前面加typedef

(例如：typedef int COUNT)

然后可以用新类型名去定义变量

(例如：COUNT I, j;)

关于typedef的一些说明

用typedef可以声明各种类型名，但不能用来定义变量。

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。

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，然后再需要用到它们的文件中用#include 命令把它们包含进来。

使用typedef 有利于程序的通用与移植。

typedef与#define 有相似之处，例如：typedef int COUNT; #define COUNT int 的作用都是用COUNT代表int。但是，它们二者是不同的。

#define是在预编译是处理的，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。

区别：typedef和define

define前面要加“#”，原因是define是命令，不用在结尾加“；”；而typedef是语句，要在结尾加“；”。

typedef (int \*) p1;

和

#define p2 int \*

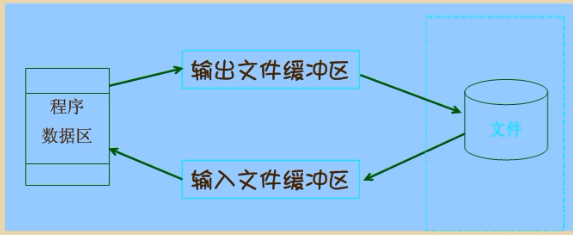
# 第十章 文件操作

## 10.1 C文件概述

所谓的“文件”是指一组相关数据的有序集合。这个数据集有一个名称，叫做文件名。实际上在前面的各章中我们已经多次使用了文件，例如源程序文件(例如：\*.c)、目标文件(例如：\*.obj)、可执行文件(例如：\*.exe 或 \*.dll)、库文件(头文件)(例如：\*.h)等。

文件通常是驻留在外部介质(如磁盘等)上的，在使用时才调入内存中来，从不同的角度可对文件作不同的分类。从用户的角度看，文件可分为普通文件和设备文件两种。

操作系统是以文件为单位对数据进行管理的。



### 10.2 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、输出设备看做是一个文件。例如：输入文件：终端键盘

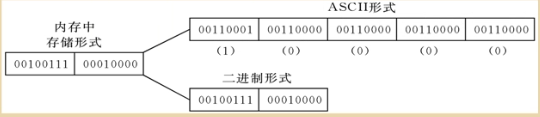
输出文件：显示屏和打印机

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内存中的存储形式原样输出到磁盘上存放

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



### 10.3 ASCII码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。

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

一般中间结果数据要暂时保存在外村上，以后有需要存入内存的，通常用二进制而建保存。

### 10.4 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。

题外话：

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的。

### 10.5 文件类型指针

文件类型指针变量：FILE \*fp；(FILE 类型定义在stdio.h中)

fp是一个指向FILE类型结构体的指针变量。我们使fp指向某一个文件的结构体变量，从而通过结构体变量中的文件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。

### 10.6 文件的打开与关闭

#### 10.6.1 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。

#### 10.6.2 文件的打开(fopen函数)

函数的调用：

FILE \*fp；

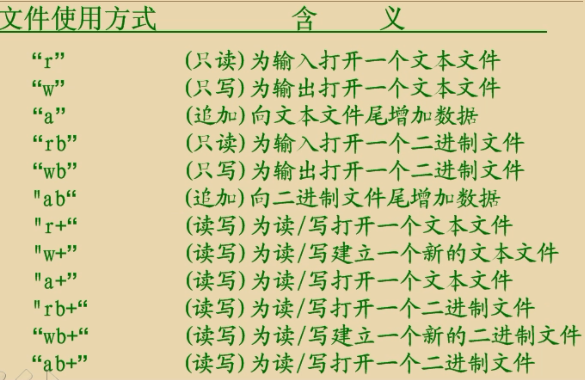
fp = fopen(文件名， 使用文件方式);

注意：

需要打开的文件名，也就是准备访问的文件的名字

使用文件的方式(“读”还是“写”等)；

让哪一个指针变量指向被打开的文件。



#### 10.6.3 对于文件使用方式有以下几点说明

凡用“r”打开一个文件时，该文件必须已经存在，且只能从该文件读出。

用“w”打开的文件只能向该文件写入。若打开的文件不存在，则以上指定的文件名建立该文件，若打开的文件已经存在，则将该文件删去，重建一个新文件。

若要向一个已经存在的文件追加新的信息，只能用“a”方式打开文件。但此时该文件必须是存在的，否则将会出错。

在打开一个文件时，如果出错，fopen将返回一个空指针值NULL。

在程序中可以用这一信息来判别是否完成打开文件的工作，并作相应的处理。

把一个文本文件读入内存时，要将ASCII码转换成二进制码，而把文件以文本方式写入磁盘时，也要把二进制码转换成ASCII码，因此文本文件的读写不存在这种转换。

示例：

#include <stdio.h>

#include <stdlib.h>

void main()

{

FILE \*fp;

if (!(fp = fopen("F:\\kang.txt", "rb")))

{

printf("Cannot open F:\\kang.txt\n");

system("pause");

}

else

{

printf("Open success!\n");

}

}

#### 10.6.4 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)

### 10.7 文件的读写

#### 10.7.1 概述

对文件的读和写是最常用的文件操作。

在C语言中提供了多种文件读写的函数：

字符读写函数：fgetc和fputc

字符串读写函数：fgets和fputs

数据块读写函数：freed和fwrite

格式化读写函数：fscanf 和fprintf

字符读写函数：fgetc和fputc

(1)字符输入输出函数(fputc()和fgetc())

fputc()函数调用：

fputc(ch, fp);

函数功能：

将字符(ch的值)输出到fp所指向的文件中去。

对于fputc函数的使用要说明几点

用写或读写方式打开一个已存在的文件时将清除原有的文件内容，写入字符从文件首开始。如需保留原有文件内容，希望写入的字符以文件末开始存放，必须以追加的方式打开文件。被写入的文件若不存在，则创建该文件。

没写入一个字符，文件内部位置指针向后移动一个字节。

fputc函数有一个返回值，如写入成功则返回写入的字符，否则返回一个EOF。可用此来判断写入是否成功。

示例：

#include <stdio.h>

#include <stdlib.h>

void main()

{

FILE \*fp;

char ch, filename[20];

printf("Please input the filename you want to write:");

scanf("%s", filename);

if (!(fp = fopen(filename, "wt+")))

{

printf("Cannot open the file!\n");

exit(0);//终止程序

}

printf("Please input the sentences you want to write:");

ch = getchar();

ch = getchar();

while (ch != EOF)//EOF = Ctrl + Z

{

fputc(ch, fp);

ch = getchar();

}

fclose(fp);

}

#### 10.7.2 字符读写函数：fgetc和fputc

fgetc()函数调用：

ch = fgetc(fp);

函数功能：

其意义是从打开的文件fp中读取一个字符并送入ch中。

对于fgetc函数的使用要说明几点

在fgetc函数调用中，读取的文件必须是以读或读写方式打开的。

在文件内部有一个指针位置指针。用来指向文件的当前读写字节。

在文件打开时，该指针总是指向文件的第一个字节。因此可连续多次使用fgetc函数，读取多个字符。

应注意文件指针和文件内部的位置指针不是一回事。

文件指针是指向整个文件的，须在程序中定义说明，只要不重新赋值，文件指针的值是不变的。

文件内部的位置指针用以指示文件内部的当前读写位置，没读写一次，该指针军向后移动，它不需要在程序中定义说明，而是由系统自动设置的。

示例：

#include <stdio.h>

#include <stdlib.h>

int main(void)

{

FILE \*fp;

char ch, filename[20];

printf("Please input the filename you want to read:");

scanf("%s", filename);

if (!(fp = fopen(filename, "r")))

{

printf("Cannot open the file!\n");

exit(0);

}

while (ch != EOF)

{

ch = fgetc(fp);

putchar(ch);

}

fclose(fp);

}

补充一下

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。

补充两下

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。

图片文件合成器：

源码：

#include <stdio.h>

#include <stdlib.h>

int main(void)

{

FILE \*f\_pic, \*f\_file, \*f\_finish;

char ch, pic\_name[20], file\_name[20], finish\_name[20];

printf("请输入需要合成的图片和文件的名称：\n");

printf("图片：");

scanf("%s", pic\_name);

printf("文件：");

scanf("%s", file\_name);

printf("生成为：");

scanf("%s", finish\_name);

if (!(f\_pic = fopen(pic\_name, "rb")))

{

printf("Cannot open this file %s.\n", pic\_name);

return;

}

if (!(f\_file = fopen(file\_name, "rb")))

{

printf("Cannot open this file %s.\n", file\_name);

return;

}

if (!(f\_finish = fopen(finish\_name, "wb")))

{

printf("Cannot open this file %s.\n", finish\_name);

return;

}

while (!(feof(f\_pic)))

{

ch = fgetc(f\_pic);

fputc(ch, f\_finish);

}

fclose(f\_pic);

while (!(feof(f\_file)))

{

ch = fgetc(f\_file);

fputc(ch, f\_finish);

}

fclose(f\_file);

fclose(f\_finish);

system("pause");

return 0;

}

#### 10.7.3 字符串的读写函数:fgets和fputs

字符串输入输出函数(fputs()和pgets())

##### 10.7.3.1 fgets函数

函数调用形式如：fgets(str, n, fp);

函数作用：

从fp所指的文件中读出n-1个字符送入字符数组str中，因为在最后加一个’\0’。

返回值：

str的首地址

##### 10.7.3.2 fputs函数

函数调用方式：fputs(“kangkang”, fp);

函数作用：其意义就是把字符串“kangkang”写入fp所指的文件之中。

返回值：

输入成功，返回值为0；

输入失败，返回值为EOF。

示例：

文件的读取：

先构造一个名字为kangkang.txt的文档文件，内容为“welcome to kangkang”

#include <stdio.h>

#include "stdlib.h"

#define LEN 256

void main(void)

{

FILE \*fp;

char buffer[LEN];

if (!(fp = fopen("kangkang.txt", "rt")))

{

printf("\nCannot open file strike any key exit!\n");

exit(1);

}

fgets(buffer, LEN, fp);

printf("%s\n", buffer);

fclose(fp);

}

文件的写入：

#include <stdio.h>

#include "stdlib.h"

#define LEN 20

void main(void)

{

FILE \*fp;

char ch, buffer[LEN];

if (!(fp = fopen("kangkang.txt", "at+")))

{

printf("\nCannot open file strike any key exit!\n");

exit(1);

}

printf("Please input a tring:\n");

fgets(buffer, LEN, stdin);//为什吗不用scanf()?stdin又是啥？

fputs(buffer, fp);

rewind(fp);//重新定义文件内部指针指到开头处

ch = fgetc(fp);

while (ch != EOF)

{

putchar(ch);

ch = fgetc(fp);

}

printf("\n");

fclose(fp);

}

/\*

scanf 这个函数用%s接收字符串的时候， 当它遇到空格回车时，

函数默认的认为字符已经接收完毕。所以，scanf函数是不能接

收字符串中的空格和回车的。如果用scanf函数，则会导致写入

的字符串不过完全。如果有一个空格或回车就会终止接收。

\*/

/\*

fgets函数是从文件获取字符串， 此时的文件是键盘(郑重的说明

一点：键盘等输入硬件也是文件。键盘是stdin文件，标准输入文

件)

\*/

#### 10.7.4 数据块读写函数(fread()和fwrinte())

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(均指其实地址)。

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:文件型的指针

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);

}

实践出真知

要求：

从键盘输入4个学生的有关数据，然后把它们以二进制的格式存储到磁盘文件中。

#include<STDIO.H>

#define SIZE 4

struct student

{

char name[10];

int num;

int age;

char addr[15];

}stu[SIZE];

void save()

{

FILE \*fp;

int i;

if (!(fp = fopen("student\_list", "wb")))

{

printf("Cannot open the file!\n");

return;

}

for (i=0; i<SIZE; i++)

{

if (fwrite(&stu[i], sizeof(struct student), 1, fp) != 1)

{

printf("File write error!\n");

fclose(fp);

}

}

}

void main()

{

int i;

printf("Please input the student's name, num, age and address:\n");

for (i=0; i<SIZE; i++)

{

scanf("%s %d %d %s", stu[i].name, &stu[i].num, &stu[i].age, &stu[i].addr);

}

save();

}

//作业：写一个load()函数将该文件读取并显示出来！

答案如下：

void load()

{

FILE \*fp;

int i;

if (!(fp = fopen("student\_list", "r")))

{

printf("Cannot open the file!\n");

return;

}

for (i=0; i<SIZE; i++)

{

fread(&stu[i], sizeof(struct student), 1, fp);

}

fclose(fp);

}

#### 10.7.5 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：

从磁盘文件中按格式读入或输出字符。

例如：

fprintf(fp, “%d, %6.2f”, i, t);

fscanf(fp, “%d, %f”, &i, &t);

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：

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

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

函数的调用形式：

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。

fseek函数应用举例：

fseek(fp, 100L, 0); 表示将位置指针移到离文件头100个字节处。

fseek(fp, 50L, 1); 表示将位置指针移到离当前位置50个字节处。

fseek(fp, 10L, 2); 表示将位置指针从文件末尾处向后退10个字节。

fseek(fp, i \* sizeof(struct stu), 0);表示将位置指针移动到离文件头i \* sizeof(struct stu)这个距离处。

举例论证

例子：

在学生文件student\_list中读出第二个学生的数据。

实现：

#include <stdio.h>

#include <stdlib.h>

struct student

{

char name[10];

int num;

int age;

char addr[15];

} boy;

int main(void)

{

FILE \*fp; //用于定位第i个结构

int i = 1;

if (!(fp = fopen("student\_list", "r")))

{

printf("Cannot open the file!\n");

return;

}

rewind(fp);

fseek(&boy, sizeof(struct student), 0);

fread(&boy, sizeof(struct student), 1, fp);

printf("name\tnumber age addr\n");

printf("%s\t%5d %7d %s\n", boy.name, boy.num, boy.age, boy.addr);

system("pause");

return 0;

}

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”);

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。

注意：

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函数的初始值自动位置为0。

clearer函数

调用形式：

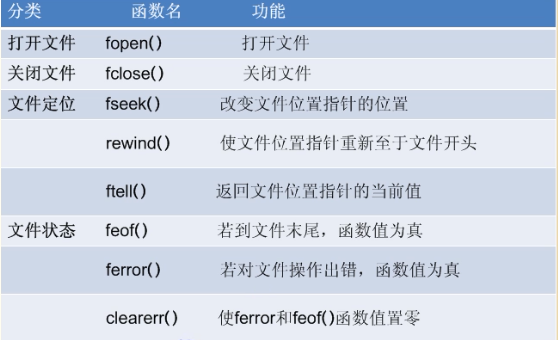
clearer(fp);

函数作用：

使文件错误标志和文件结束标志值为0。

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

#### 10.7.6 文件输入输出小结





# 第十一章 位运算

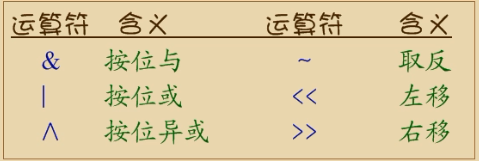
## 1位运算符和位运算

### 1.1概念

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。

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。

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。



(1)位运算符中除~以外，均为二目(元)运算符，即要求两侧各有一个运算量。

(2)运算量只能是整型或字符型的数据，不能为实型数据。

## 2 “按位与”运算符(&)

### 2.1 含义

参加运算的两个数据，按二进制位进行“与”运算。

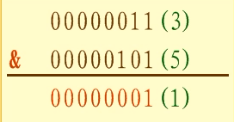
如果两个相应的二进制位都为1，则该位的结果值为1；否则为0。

例如：

0 & 0 = 0， 0 & 1 = 0， 1 & 0 = 0， 1 & 1 = 1.

注意：

3&5并不等于8，应该是按位与运算：



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

### 2.2 按位与运算的用途

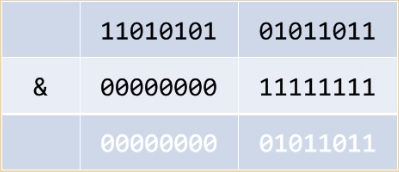
#### 2.2.1 清零

若想对一个存储单元清零，即使其全部二进制位为0，只要找一个二进制数，其中各个位符合以下条件：原来的数中为1的位，新数中相应位0.然后使二者进行&运算，即可达到清零的目的。

例如：要求将二进制数11100101的第2位清零(第2位是从右向左的第3位)

#### 2.2.2 取一个数的某些指定位

例如：我们需要对一个字形数据取出其第8位的值时，我们可以这么做。

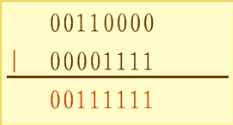


## 3 “按位或”运算符(|)

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.

例如：



题目：

编写一个小程序，将输入的大写字母转换为小写字母，输入的小写字母转换为大写字母，要求用位操作完成转换过程。

提示：区分大小写的地方在于字母ASCII码对应的二进制代码中的第5位(从右向左第6位)。大写字母第5位是0，而小写字母第5位是1.

实现：

#include <STDIO.H>

int main(void)

{

char ch, temp;

printf("请输入一个字母:\n");

ch = getchar();

temp = getchar();

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

{

printf("输入有误，请重新输入一个字母!\n");

ch = getchar();

}

if (ch & 32)

{

ch = ch & 223;

}

else

{

ch = ch | 255;

}

putchar(ch);

ch = getchar();

putchar(ch);

printf("\n");

return 0;

}

## 4 “异或”运算符(^)

### 4.1 概念

异或运算符^也称为XOR运算符。

它的规则是：

若参加运算的两个二进制同号则结果为0(假)， 异号则结果为1(真)

即：0^0 = 0, 0^1 = 1, 1^1 = 1, 1^0 = 1.

例如：



### 4.2 ^运算符应用

#### 4.2.1 使特定位翻转

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



#### 4.2.2 与0相^，保留原值

例如：



因为原数中的1与0进行运算得1,0^0得0，故保留原数。

#### 4.2.3 交换两个值，不用临时变量

例如：a = 3, b = 4, 现在想将a、b变量的值交换位置，我们传统的做法是定义多一个temp变量，而现在temp去度蜜月了，怎么办？

我们可以这样做：

a = a^b;

b = b^a;

a = a^b;

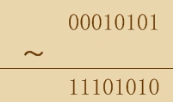
这种方法也长应用于加密算法。

## 5 “取反”运算符(~)

~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)按位取反。

例如：



## 6 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 = <<2 将a 的二进制数左移2位，右边补0.

若a = 15,即二进制数00001111，左移2位得00111100，(十进制数60)

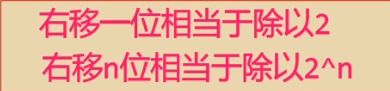
若高位左移后溢出，舍弃。

左移1位相当于该数乘以2，左移2位相当于该数乘以4，15<<2 = 60，即乘以了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。

假设以一个字节(8位)存一个整数，若a为无符号整型变量，则a = 64时，左移一位时溢出的是0，而左移2位时，溢出的最高位包含1.

例如：a = 017时：a的值用二进制形式表示为00001111，舍弃第2位11，得到a>>2 = 00000011



在右移时，需要注意符号位问题：

对无符号数，右移时左边高位移入0；

对于有符号的值，如果原来符号位为0(该数为正)，则左边也是移入0.

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，有点系统移入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”。

例如：a的值是十进制数-2；

a == 1111 1110(用二进制形式表示)

无符号数：a >> 1: 0111 1111(逻辑右移时)

有符号数： a>> 1: 1111 1111(算术右移时)

示例：

#include <STDIO.H>

int main(void)

{

signed char a = -2;//同时尝试unsigned

a = a >> 1;

printf("%d\n\n", a);

return 0;

}

## 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：

&=, |=, >>=, <<=, ^=

所以，a &= b相当于a = a & b

a <<= 2 相当于a = a << 2

## 8 位运算举例

取一个char a从右端开始的2~5位。

1.先使a右移2位：a>>2

目的是使要取出的那几位移到最右段，图示：

2.设置一个第4位全为1，其余全为0的数。

~(~0 << 4)

3.将上面1.、2.进行&运算。

(a >> 4) & ~(~0 << 4)

编程检验

#include <stdio.h>

int main(void)

{

char a, b, c, d;

printf("请输入待检验数字:\n");

scanf("%d", &a);

b = a >> 2;

c = ~(~0 << 4);

d = b & c;

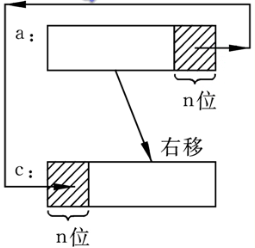
printf("%d\n\n", d);

return 0;

}

循环移位

要求将a进行右循环移位



实现：

#include <stdio.h>

#include <STDLIB.H>

int main(void)

{

unsigned char a, b, c;

int n;

printf("请输入需要实现循环右移的数字:\n");

scanf("%d", &a);

printf("请输入需要实现右移的位数:\n");

scanf("%d", &n);

b = a << (sizeof(char) \* 8 - n);

c = a >> n;

c = c | b;

printf("结果是:%d\n", c);

system("pause");

return 0;

}

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。

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。

struct packed\_data

{

unsigned a:2;

unsigned b:6;

unsigned c:4;

unsigned d:4;

}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:2;另一个存储单元

a、b、c应该连续存放在一个存储单元中，由于用了长度为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，视不同的编译系统而异)。

(3)一个位段必须存储在以同一个存储单元中，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)位段可以再数值表达式中引用，它会被系统自动地转换成整型数。

# 附录

判断星期几的程序，很累的。这段程序存在很多问题。没有用到代码的模块化。代码很乱。不过功能还可以。算法很简单。

#include <stdio.h>

#include <math.h>

#include <conio.h>

void main()

{

int year, month, day, week, date;

int y, m, d;

int w;

char a;

printf("====================================\n本程序是用来测试星期以及平年还是闰年。\n 小康制作\n\n");

while(1)

{printf("\n按1开始运行程序，按0退出。\n");

a = getch();

putch(a);

if ('0' == a)

{

break;

}

if ('1' == a)

{

printf("\n开始\n请输入年月日，按空格分开。\n\n");

scanf("%d%d%d", &year, &month, &day);

if (0 > year || 0 > month || 0 > day)

{

printf("输入错误\n");

return;

}

if (month > 12 || day > 31)

{

printf("输入错误\n");

return;

}

switch(month)

{

case 4:

case 6:

case 9:

case 11:

if (day > 30)

{

printf("输入错误\n");

}

}

if (year %100 == 0)

{

if(year % 400 == 0)

{

printf("该年份为闰年。\n");

if (2 == month && day > 29)

{

printf("输入错误\n");

return;

}

y = fabs((year - 2012)/4) + fabs((year - 2012) \* 365);

switch(month)

{

case 1:m = 0;break;

case 2:m = 31;break;

case 3:m = 31+29;break;

case 4:m = 62+29;break;

case 5:m = 92+29;break;

case 6:m = 123+29;break;

case 7:m = 153+29;break;

case 8:m = 184+29;break;

case 9:m = 215+29;break;

case 10:m = 245+29;break;

case 11:m = 276+29;break;

case 12:m = 306+29;break;

}

d = day;

date = y + m + d;

w = (date - 3) % 7;

}

else

{

printf("该年份为平年。\n");

if (2 == month && day > 28)

{

printf("输入错误\n");

return;

}

y = fabs((year - 2012)/4) + fabs((year - 2012) \* 365);

switch(month)

{

case 1:m = 0;break;

case 2:m = 31;break;

case 3:m = 31+28;break;

case 4:m = 62+28;break;

case 5:m = 92+28;break;

case 6:m = 123+28;break;

case 7:m = 153+28;break;

case 8:m = 184+28;break;

case 9:m = 215+28;break;

case 10:m = 245+28;break;

case 11:m = 276+28;break;

case 12:m = 306+28;break;

}

d = day;

date = y + m + d;

w = (date - 3) % 7;

}

}

else if (year % 4 == 0)

{

printf("该年份为闰年。\n");

if (2 == month && day > 29)

{

printf("输入错误\n");

return;

}

y = fabs((year - 2012)/4) + fabs((year - 2012) \* 365);

switch(month)

{

case 1:m = 0;break;

case 2:m = 31;break;

case 3:m = 31+29;break;

case 4:m = 62+29;break;

case 5:m = 92+29;break;

case 6:m = 123+29;break;

case 7:m = 153+29;break;

case 8:m = 184+29;break;

case 9:m = 215+29;break;

case 10:m = 245+29;break;

case 11:m = 276+29;break;

case 12:m = 306+29;break;

}

d = day;

date = y + m + d;

if (year >= 2012)

{

w = (date - 1) % 7;

}

if (year < 2012)

{

w = (date - 4) % 7;

}

}

else

{

printf("该年份为平年。\n");

if (2 == month && day > 28)

{

printf("输入错误\n");

return;

}

y = fabs((year - 2012)/4) + fabs((year - 2012) \* 365);

switch(month)

{

case 1:m = 0;break;

case 2:m = 31;break;

case 3:m = 31+28;break;

case 4:m = 62+28;break;

case 5:m = 92+28;break;

case 6:m = 123+28;break;

case 7:m = 153+28;break;

case 8:m = 184+28;break;

case 9:m = 215+28;break;

case 10:m = 245+28;break;

case 11:m = 276+28;break;

case 12:m = 306+28;break;

}

d = day;

date = y + m + d;

if (year > 2012)

{

w = date % 7;

}

if (year < 2012)

{

w = (date - 3) % 7;

}

}

switch(w)

{

case 1:printf("星期一\n");break;

case 2:printf("星期二\n");break;

case 3:printf("星期三\n");break;

case 4:printf("星期四\n");break;

case 5:printf("星期五\n");break;

case 6:printf("星期六\n");break;

case 0:printf("星期日\n");break;

}

}

}

}

//2012年1月1日为星期日