



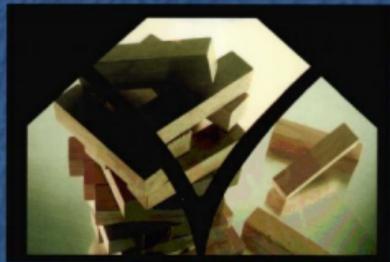
· 全国高职高专电子信息类系列规划教材 ·

C语言程序设计

项目化教程

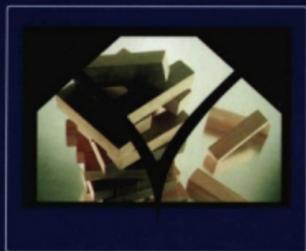
C Yuyan Chengxu Sheji Xiangmuhua Jiaocheng

● 陈兴无 编著



华中科技大学出版社

<http://www.hustp.com>



- 策划编辑：张毅
- 责任编辑：张毅
- 封面设计：刘卉

ISBN 978-7-5609-5137-9



9 787560 951379 >

定价：29.00元



· 全国高职高专电子信息类系列规划教材 ·

C语言程序设计

项目化教程

C Yuyan Chengxu Sheji Xiangmuhua Jiaocheng

陈兴无 编著



华中科技大学出版社
中国·武汉

图书在版编目(CIP)数据

C语言程序设计项目化教程/陈兴无 编著. —武汉:华中科技大学出版社, 2009年3月

ISBN 978-7-5609-5137-9

I. C… II. 陈… III. C语言-程序设计-教材 IV. TP312

中国版本图书馆 CIP 数据核字(2009)第 025072 号

C语言程序设计项目化教程

陈兴无 编著

策划编辑:张毅

责任编辑:张毅

责任校对:刘峻

封面设计:刘卉

责任监印:周治超

出版发行:华中科技大学出版社(中国·武汉)

武昌喻家山 邮编:430074 电话:(027)87542624

录排:武汉正风图文照排中心

印刷:武汉市新华印刷有限责任公司

开本:787mm×1092mm 1/16 印张:18

版次:2009年3月第1版

印次:2009年3月第1次印刷

字数:420 000

定价:29.00元

ISBN 978-7-5609-5137-9/TP·675

(本书若有印装质量问题,请向出版社发行科调换)

前言

目前,高职院校的计算机专业及其相关专业大都将C语言程序设计作为一门重要的基础课程。在多年的教学实践中,我们体会到要真正掌握C语言,学习者的难度较大。一方面是课程自身有一定的难度,另一方面是现有教材不能很好地将教学过程中出现的知识、技能与实际软件开发所需要的知识、技能结合起来,学不能用、学不为用,学习者的积极性和主动性不能得到充分发挥。我们本着“职业活动导向、任务驱动、项目载体”的教学原则,结合C语言程序设计自身的特点,用设计独特、编排新颖、通俗易懂的方法编写了本书。

本书具有以下几个特点。

本书以完成“班级学生成绩管理系统”软件开发为项目,系统地描述了软件开发的全过程。为了更好地完成该项目,我们将其分解成15个任务,合理地安排到相关单元中,做到开发项目需要什么知识就讲解什么知识,并做适当地知识扩展。摒弃了以往学习结束后采用集中实训来完成的方式。

之所以选择“班级学生成绩管理系统”作为贯穿全书的项目,其原因如下。

(1) 项目本身不陌生,有一种亲切感

软件开发应当由专业工程师和程序开发工程师共同完成。读者要开发某个软件时,他实际上同时承担了这两种职责。一个熟悉的项目,有利于读者不用过多地纠缠专业细节,有利于读者专注于开发程序相关知识的学习。

(2) 项目不宜过大、过繁

过大的项目不适应初学者。一个真正的学生成绩管理系统软件,包含的内容十分丰富,涉及面太广,读者一时难以抓住课程的精髓。我们在项目名前加上“班级”二字,其目的是简化软件功能,有利于基础知识与基本技能的学习。因此,我们开发的是一个“学习型”的软件。

(3) 知识的覆盖面能满足学习的需要

第一,“班级学生成绩管理系统”虽然是学习型的软件,但它还是包含了学习 C 语言程序设计的大部分知识和技能点,只有少部分内容不会涉及。我们将暂时没有涉及的内容安排在“扩展知识与理论”一节中,可供自由选择。

第二,C 语言教科书中也有把学生成绩管理系统作为实践内容的,但它们大多是安排在最后,作为集中实训项目;有的教科书用某个项目来贯穿整个教学过程,但它们较多地改变了 C 语言前后关联的知识结构,这都不利于初学者学习。合理地安排各单元任务,尽量保证知识本来的结构体系,能极大地提高读者的学习兴趣,提高学习的积极性,达到事半功倍的效果。

第三,本书在每个单元中都安排了与本单元有关的能力训练任务,其目的是帮助读者更好地完成项目。

第四,本书安排了大量的实例。大量的实例有利于读者接触各种程序设计方法,有些实例还用不同的方法进行了多次设计,这样做会使读者更好地了解和掌握程序开发的灵活性。全部实例都用 VC++6.0 调试通过。

第五,本书结合编者多年的教学实践,在除第一单元之外的其他各单元中增加了初学者常见错误及处理方法一节,其目的就是想使读者少走弯路,尽快掌握程序开发方法。

第六,本书每单元结束后都安排了一定量的实训内容,便于读者巩固所学的知识。

第七,本书的教学课件、深入训练答案和项目程序,可在 <http://www.eszy.edu.cn> 网站中下载。

段昌盛、文晓华、杨平华、高寿斌四位同志参加了本书编写大纲的讨论,并提出了许多宝贵意见,在此表示感谢。

本书在编写过程中,还得到了恩施职业技术学院领导、教务处和计算机与信息工程系的大力支持,在此一并表示衷心的感谢。

由于水平和能力有限,本书难免有不当之处,恳请读者多加批评指正。

编者
2009 年元月

数字知识
PDG

目 录

单元 1 系统设计

1.1 任务 1:“班级学生成绩管理系统”总体规划设计	(1)
1.2 必备知识与理论	(3)
1.2.1 C 语言的特点	(3)
1.2.2 几个简单的 C 语言程序	(4)
1.2.3 VC++6.0 开发工具介绍	(6)
1.3 扩展知识与理论	(10)
1.3.1 算法的概念	(11)
1.3.2 算法的描述方法	(11)
1.4 深入训练	(12)
习题 1	(13)

单元 2 项目数据设计与数据运算

2.1 任务 2:“班级学生成绩管理系统”中相关数据设计	(14)
2.2 必备知识与理论	(15)
2.2.1 数据类型概述	(15)
2.2.2 常量与变量	(16)
2.2.3 简单数据类型	(17)
2.2.4 数据运算符及其表达式	(23)
2.2.5 不同数值型数据间的混合运算	(31)
2.3 扩展知识与理论	(33)
2.3.1 位运算符和位运算	(33)
2.3.2 常见错误及处理方法	(36)
2.4 深入训练	(37)
习题 2	(37)

单元 3 项目封面与菜单的初步设计

- 3.1 任务 3:用输入/输出函数初步设计项目封面与菜单····· (41)
- 3.2 必备知识与理论····· (45)
 - 3.2.1 C 语句····· (45)
 - 3.2.2 格式化输入/输出函数····· (46)
- 3.3 扩展知识与理论····· (53)
 - 3.3.1 单个字符输入/输出函数····· (53)
 - 3.3.2 常见错误及处理方法····· (55)
- 3.4 深入训练····· (55)
- 习题 3····· (56)

单元 4 项目封面、菜单的顺序执行设计

- 4.1 任务 4:项目封面、菜单的顺序执行设计····· (59)
- 4.2 必备知识与理论····· (61)
 - 4.2.1 顺序结构程序设计····· (61)
 - 4.2.2 顺序结构特点····· (62)
- 4.3 深入训练····· (62)
- 习题 4····· (63)

单元 5 项目菜单的选择执行设计

- 5.1 任务 5:用 if 语句实现菜单的选择执行设计····· (66)
- 5.2 任务 6:用 switch 语句实现菜单的选择执行设计····· (68)
- 5.3 必备知识与理论····· (69)
 - 5.3.1 if 语句和条件运算····· (69)
 - 5.3.2 switch 语句····· (74)
- 5.4 常见错误及处理方法····· (78)
- 5.5 深入训练····· (78)
- 习题 5····· (79)

单元 6 项目菜单的循环选择执行设计

- 6.1 任务 7:用循环语句实现项目主菜单的选择执行设计····· (82)
- 6.2 任务 8:用循环语句实现项目主、子菜单的选择执行设计····· (85)
- 6.3 必备知识与理论····· (88)
 - 6.3.1 for 语句····· (89)
 - 6.3.2 while 语句····· (91)
 - 6.3.3 do-while 语句····· (92)
 - 6.3.4 循环的嵌套····· (93)
 - 6.3.5 break 语句与 continue 语句····· (95)
- 6.4 扩展知识与理论····· (97)
 - 6.4.1 良好的源程序书写习惯····· (97)

6.4.2 常见错误及处理方法	(99)
6.5 深入训练	(100)
习题6	(100)
单元7 项目的整体框架设计	
7.1 任务9:项目的整体框架设计	(103)
7.2 必备知识与理论	(109)
7.2.1 结构化程序设计思想与函数分类	(109)
7.2.2 函数的定义与调用	(110)
7.2.3 函数的嵌套调用和递归调用	(114)
7.2.4 函数调用中的参数传递	(117)
7.3 扩展知识与理论	(119)
7.3.1 变量的作用域	(119)
7.3.2 变量的生存期	(123)
7.3.3 预处理命令	(126)
7.3.4 常见错误及处理方法	(131)
7.4 深入训练	(132)
习题7	(133)
单元8 项目中数组的应用	
8.1 任务10:初步完善学生最高、最低等成绩查找	(136)
8.2 任务11:初步完善学生成绩排序	(140)
8.3 必备知识与理论	(142)
8.3.1 数组概述	(142)
8.3.2 一维数组的定义及其应用	(142)
8.3.3 数组作函数参数	(145)
8.3.4 字符数组的定义及其应用	(150)
8.4 扩展知识与理论	(157)
8.4.1 二维数组的定义及其应用	(157)
8.4.2 常见错误及处理方法	(162)
8.5 深入训练	(163)
习题8	(163)
单元9 项目中指针的应用	
9.1 任务12:用指针实现学生最高、最低等成绩查找	(166)
9.2 任务13:用指针实现学生成绩排序	(169)
9.3 必备知识与理论	(170)
9.3.1 内存地址与数据指针的概念	(170)
9.3.2 指向变量的指针变量	(172)
9.3.3 数组指针和指向数组的指针变量	(181)
9.3.4 字符串的指针访问法	(186)
9.4 扩展知识与理论	(190)

9.4.1 二维数组元素的指针访问方式	(190)
9.4.2 指针数组与带参数的 main 函数	(193)
9.4.3 常见错误及处理方法	(198)
9.5 深入训练	(200)
习题 9	(200)

单元 10 项目中结构体的应用

10.1 任务 14:用结构体实现数据的增加、删除、修改和显示	(203)
10.2 必备知识与理论	(210)
10.2.1 结构体概述	(210)
10.2.2 结构体类型的应用	(211)
10.2.3 结构体数组的应用	(223)
10.3 扩展知识与理论	(229)
10.3.1 结构体变量作函数类型	(229)
10.3.2 共用体类型的应用	(231)
10.3.3 枚举类型的应用	(238)
10.3.4 常见错误及处理方法	(240)
10.4 深入训练	(241)
习题 10	(242)

单元 11 项目中中学生数据的存储与重用

11.1 任务 15:项目中中学生数据的存储和重复使用	(245)
11.2 必备知识与理论	(247)
11.2.1 文件的概念	(247)
11.2.2 文件的基本操作	(248)
11.2.3 文件的定位	(257)
11.3 扩展知识与理论	(261)
11.3.1 读/写字符串和格式化读/写数据函数	(261)
11.3.2 文件状态检测	(267)
11.3.3 常见错误及处理方法	(268)
11.4 深入训练	(269)
习题 11	(269)
附录	(274)
参考文献	(279)



单元1 系统设计

能力目标

- 学会与人打交道,完成调查任务。
- 能够启动 Visual C++ 6.0,并能正确进入编程窗口。
- 能根据实例完成三种类型的简单 C 语言程序编写。
- 能初步掌握“班级学生成绩管理系统”工作模块构成。

知识目标

- 了解 C 语言的特点和 C 语言程序的开发步骤。
- 初步了解 C 程序的组成结构和主函数的作用。
- 初步掌握 C 语言流程图和 N-S 图的图例特点与属性。

学习提示

本单元主要介绍“班级学生成绩管理系统”的设计要点和主要模块,介绍 C 语言程序设计中的一般基本概念以及开发应用程序的主要步骤。通过本单元的学习,使读者对用 C 语言开发程序有一个概括性的了解,并能够模仿例题编写一些简单程序。

1.1 任务 1:“班级学生成绩管理系统”总体规划设计

为了使学习者对开发应用软件有较为系统的了解,我们设计了一个贯穿整个教学过程的程序设计任务——班级学生成绩管理系统。之所以在“学生成绩管理系统”前加上“班级”二字,就是为了简化学习的难度,将学习的注意力放在主要功能的实现上。

学生成绩统计与管理,是学校教学管理中的重要管理内容,它关系到学生是否能正常毕业。随着学校规模的扩大和管理要求的提高,传统的手工管理方法就不适应当前学校教学管理的需要。另外,学生学习成绩的计算机自动化管理也是衡量一个学校管理水平高低的标志,它能高效、方便地管理学生成绩。

要开发一个班级学生成绩管理系统软件,可以先走访本校的教务管理部门,了解学生

成绩管理方法,经过分析得出管理流程,按管理流程设计出管理模块。如果是正式开发管理软件,这个工作要经过与用户单位充分的讨论、论证,最后得出一致的意见。

下面就学习型的“班级学生成绩管理系统”的主要功能进行说明。

“班级学生成绩管理系统”共设计了六大功能模块。

(1) 打开文件模块,能够打开保存在磁盘上的学生成绩文件。

(2) 保存文件模块,能将一个班 40 个学生的学号、姓名、三门课程的学习成绩和总成绩以及平均成绩全部保存在磁盘文件中。

(3) 编辑成绩模块,能编辑学生信息和学生成绩,并能进行相应的增加、删除、修改等操作。

(4) 显示成绩模块,能显示全部学生信息、指定学生信息、不及格学生信息和按总成绩排序后的学生信息。

(5) 计算模块,能对学生成绩进行总成绩与平均成绩计算,能找出全班学习成绩最好的和最差的学生。

(6) 程序说明模块,能对软件的版本、功能、使用方法、开发者信息进行相应说明。

另外,本系统只有一个出口,程序只能通过该出口正常结束,以保证安全退出系统。六大功能模块可以用图 1.1 表示。



图 1.1 “班级学生成绩管理系统”功能模块示意图

开发“班级学生成绩管理系统”项目遵循“软件开发工作流程”和“循序渐进”原则,分任务实施。本书各单元首先以一至两个任务为驱动,围绕完成任务设计必备知识与理论,力争程序开发步骤与知识水平、能力紧密结合,使学习与应用融为一体,学用结合,学用相长。通过学习的深入逐步完善程序功能,最后形成一个完整的程序。随着知识的逐渐增

多,学习者也可以自己增加新模块,使程序更趋完善,更加实用。

项目开发实施方案分别如下。

任务 1:“班级学生成绩管理系统”总体规划设计。

任务 2:“班级学生成绩管理系统”中相关数据设计。

任务 3:用输入/输出函数初步设计项目封面与菜单。

任务 4:项目封面、菜单的顺序执行设计。

任务 5:用 if 语句实现菜单的选择执行设计。

任务 6:用 switch 语句实现菜单的选择执行设计。

任务 7:用循环语句实现项目主菜单的选择执行设计。

任务 8:用循环语句实现项目主、子菜单的选择执行设计。

任务 9:项目整体的框架设计。

任务 10:初步完善学生最高、最低等成绩查找。

任务 11:初步完善学生成绩排序。

任务 12:用指针实现学生最高、最低等成绩查找。

任务 13:用指针实现学生成绩排序。

任务 14:用结构体实现数据的增加、删除、修改和显示。

任务 15:项目中学生数据的存储和重复使用。

单元能力训练任务分别如下。

任务 A:调查了解本校学生成绩管理的工作流程。

任务 B:画出本校学生成绩管理工作流程模块图。

任务 C:正确安装 Visual C++ 6.0 软件。

任务 D:模仿任务 A 设计“学生通讯录”工作流程模块图。

1.2 必备知识与理论

C 语言是一种中级语言,它既具有高级语言便于学习与开发,接近人类自然语言的特点,又具有低级语言能直接操作计算机硬件的特点。这两大特点使它成为开发系统软件和应用软件强有力的工具。C 语言是目前国际上广泛流行的一种结构化的程序设计语言。

1.2.1 C 语言的特点

C 语言与其他许多语言相比,其特点如下。

(1) C 语言总共只有 32 个关键字(见附录 II),9 种控制语句(见附录 III),语言简洁、紧凑,使用方便、灵活。

(2) C 语言共有 34 种运算符(见附录 IV)。运算符丰富,由运算符组成的表达式类型多样化。(一般而言,学习 C 语言的过程就是学习掌握关键字、控制语句和运算符的过程。)

(3) C 语言的类型丰富,既有系统定义的简单类型,如整型、实型、字符型等,又有用户自定义的构造类型,如数组类型、结构体类型、共用体类型等。

(4) C语言利用三种简单的控制结构(顺序结构、分支结构、循环结构)就能实现任何复杂结构。采用函数作为程序的模块单位,便于实现程序的模块化。

(5) C语言语法限制不太严格,程序设计自由度大。

(6) C语言生成目标代码质量高,一般只比汇编语言生成的目标代码效率低10%~20%。

(7) 用C语言编写的程序可移植性好。一般不用修改就能用于各种型号的计算机和各种操作系统。

(8) C语言学习难度较大,特别是函数、指针、地址等内容难度大,需要认真学习才能掌握。

尽管C语言没有其他语言好掌握,但对编写系统软件和应用软件,用C语言明显优于其他语言。它还是后续课程(如数据结构、软件工程)的先修课,所以C语言是一种十分优秀而又十分重要的语言,如果想成为一名优秀的软件工程师,就必须认真地学好C语言。

1.2.2 几个简单的C语言程序

为了使学习者对C语言有一个初步认识,下面先通过三个简单实例来分析C语言程序的结构。

【例 1.1】 一个最简单的C语言程序,在屏幕上显示“Hello,world”。

```
1 #include <stdio.h>
2 main() /*主函数*/
3 {
4     printf("Hello,world\n"); /*输出函数*/
5 }
```

程序运行结果:

```
Hello,world
```

这是世界上第一个C语言程序,整个程序只有一个主函数(main),主函数中只有一条输出语句,这条语句放在一对大括号中,它的功能是在系统默认的输出设备(显示器)上输出“Hello,world”,实现这一功能是使用系统提供的标准库函数printf()。

注意:为了方便理解和教学,给程序加了行号,但在上机调试时不能带行号。

【例 1.2】 求两数之和。

```
1 #include <stdio.h>
2 main()
3 {
4     int a,b,sum; /*定义了三个整型变量*/
5     a=1234;
6     b=5678;
7     sum=a+b;
8     printf("sum=%d\n",sum); //打印 a 和 b 之和
9 }
```

程序运行结果：

```
sum=6912
```

本程序的作用是求两个整数 a 和 b 之和。第 1 行是文件包含命令行；第 2 行 main 表示主函数，函数语句包含在一对大括号中；第 4 行定义三个变量；第 5 行和第 6 行分别给变量 a 和 b 赋值；第 7 行将 a、b 之和赋给 sum；第 8 行使用输出函数输出两数之和。另外，/*、*/ 之间和// 之后的文字分别表示注释，为便于理解，用汉字进行注释。

【例 1.3】 从键盘上输入两个整数，比较两数，将大的数输出。

```
1 int max (int x,int y)//定义 max 函数,形式参数 x,y 为整型
2 {
3     int z; /*max 函数中的声明部分,定义变量 z 为整型*/
4     if(x>y)z=x;
5     else z=y;
6     return(z); /*将 z 的值返回,通过函数调用将值带回到调用处*/
7 }
8 void main ( )
9 {
10    int max(int x,int y);      //函数声明
11    int a,b,c;
12    printf("输入两个整型数据:");/*输入提示*/
13    scanf("%d,%d",&a,&b);      /*输入变量 a 和 b 的值*/
14    c=max(a,b);                /*调用 max 函数,将返回的值赋给 c*/
15    printf("max=%d",c);       /*输出 c 的值*/
16 }
```

程序运行结果：

输入两个整型数据：5,9<回车>（注：本书中用户输入的部分均添加下划线）

```
max=9
```

本程序包括两个函数：主函数和被调用的函数 max。max 函数的作用是比较 x 和 y 的值，并将较大者赋给变量 z。return 语句将 z 的值返回给主调函数 main。main 函数中的 scanf() 是系统提供的标准库函数，其功能是输入 a 和 b 的值，&a 和 &b 中的“&”的含义是“取地址”，即将两个输入的数值分别赋给变量 a 和变量 b 的地址所标识的单元中，也就是将两数值输入给变量 a 和 b，程序第 14 行的功能为调用 max 函数。

综合上述三个例题，可以得到 C 语言程序的基本特点如下。

(1) C 语言程序是由函数构成的。它可以由一个主函数(main)组成，也可以由一个 main 函数和若干个其他函数组成，一个 C 语言源程序有一个，而且只有一个 main 函数。因此，函数是 C 语言程序的基本单位。被调用的函数可以是系统提供的函数(如 printf() 和 scanf())，也可以是用户根据需要自己设计的函数(如 max())。程序的全部工作都是由各个函数来完成的，编写 C 语言程序就是编写一个个函数。

(2) 一个函数由以下两部分组成。

① 函数的首部，即函数的第一行。包括函数名、函数类型、函数属性、函数参数名、参

数类型等。一个函数名后面必须跟一对圆括号,这也是函数的标志,函数参数可以没有,如 `main()`。

② 函数体,即函数首部下面的大括号 `{ }` 内的部分。如果一个函数内有多个大括号,则最外层的一对 `{ }` 为函数体的范围。

函数体包括以下两部分。

声明部分:声明所用到的变量或声明所调用的函数。例如,例 1.2 中第 4 行,例 1.3 中第 3、10、11 行。

执行部分:由若干语句组成。例如,例 1.2 中第 5 行至第 8 行。

当然,有时也可以没有声明部分,甚至既没有声明部分,也没有执行部分。例如,

```
dump() { }
```

这是一个空函数,什么都不做,但它是合法的,这样的函数主要起占位作用。

(3) 不论 `main` 函数在程序中的什么位置,C 语言程序总是从 `main` 函数开始执行。一般而言,`main` 函数执行完后程序也就结束了。也就是说,`main` 函数是程序的入口和出口。

(4) C 语言程序书写格式自由,一行内可以有几条语句,一条语句也可以写成多行。C 语言程序没有行号,但每条语句和数据定义的最后必须有一个分号。分号是 C 语言语句的必要组成部分。

(5) C 语言本身没有输入/输出语句。输入/输出是通过库函数 `scanf()` 和 `printf()` 等函数来完成的。

(6) C 语言可以用 `/* */`(块注释)对 C 语言程序中的任何部分作注释,VC++ 中还可以用 `//`(行注释)来给程序加注释,两者的区别在于 `/* */` 可以对多行进行注释,而 `//` 只能对单行进行注释。注释部分是不会参加编译的。

1.2.3 VC++ 6.0 开发工具介绍

1. C 语言程序上机调试步骤

(1) 编辑。编辑源程序,一般用文字处理软件编写,当然也可以用集成化的程序设计软件,其中包括了文字处理部分。C 语言源程序的扩展名为 `.C`。

(2) 编译。源程序编写好之后,可以进行编译。编译是将源程序转换成二进制文件,即目标文件,扩展名为 `.obj`(注意:源程序中的注释是不会被编译的)。在编译过程中,将发现在源程序编写过程中出现的错误,这种错误一般是由书写错误造成的,因此,这种错误我们形象地称为语法错误,这种错误是易于修改的。

(3) 链接。编译成功后的文件并不能运行,因为这种程序虽然称为目标文件,但仍是半成品,不能执行。在目标程序中还没有为函数、变量等安排具体的地址,因此也称为浮动程序。所以链接就是将若干目标文件加以归并、整理,为所有的函数、变量分配具体的地址,同时将库函数链接到 `.obj` 文件中,生成可执行的文件,扩展名为 `.exe`。

在链接的过程中也可能发现错误,这种错误是由设计不足或缺陷引起的,一般不易发现,我们称这种错误为逻辑错误。

(4) 运行。根据运行的不同目的,运行可分为应用运行、测试运行和调试运行。

① 应用运行是指程序正式投入使用后的运行,目的是通过程序运行完成预先设定的功能,从而获得相应的效益。

② 测试运行是应用运行前的试运行,是为了验证整个应用系统的正确性,如果发现错误,应进一步判断错误的原因和产生错误的大致位置,以便加以纠正。

③ 调试运行是专门为验证某些函数的正确性而进行的,被运行的主函数通常就是一个调试程序。运行时,通过输入一些特定的数据,观察它是否产生预期的输出结果。如果发现任何不正常的情况,应配合使用程序跟踪等手段,观察程序是否按预期的流程运行,程序中的某些变量的值是否如预期的那样变化,从而判定出错的具体原因和位置,以便加以纠正。

上机调试程序步骤如图 1.2 所示。

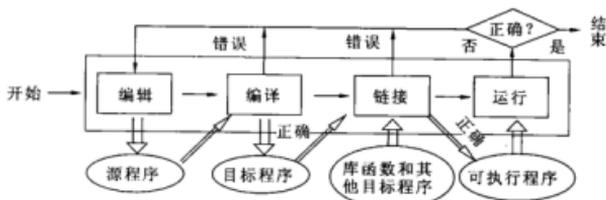


图 1.2 上机调试程序步骤

2. 开发工具介绍

1) 启动 VC++ 程序

在“开始”菜单中选择“程序”菜单,再选择“Microsoft Visual Studio 6.0”菜单,在下拉菜单中点击“Microsoft Visual C++ 6.0”,如图 1.3 所示。

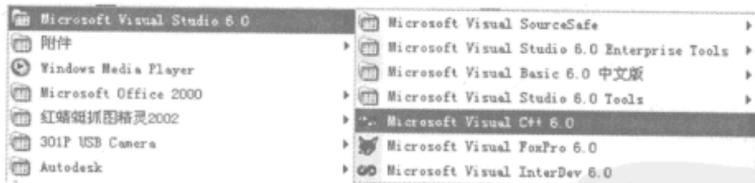


图 1.3 打开 VC++ 6.0

随后进入 VC++ 编辑窗口,如图 1.4 所示。

VC++ 编辑窗口和一般的 Windows 窗口并无太大的区别。它由标题栏、菜单栏、工具栏、工作区、编辑区、调试信息显示区和状态栏组成。在没有编辑文件的情况下,工作区无信息显示,编辑区为深灰色。

2) 菜单栏和工具栏

由于 VC++ 能够编辑 C++ 语言程序,而 C++ 语言程序又是 C 语言的超集,功能



图 1.4 VC++ 编辑调试窗口

比 C 语言强大得多,因此这里对菜单栏和工具栏只介绍一小部分,其余部分留给读者自己去学习。

(1) 菜单栏。

VC++ 程序设计开发工具共有九个菜单,它们分别是 File、Edit、View、Insert、Project、Build、Tools、Window 和 Help。在学习中将会遇到 File 和 Build 菜单下的部分子菜单,现介绍如下。

“File”菜单用于文件的相关操作,如图 1.5 所示。各子菜单的中文意思如下。



图 1.5 “File”菜单

- ① New:新建文件。
- ② Open:打开已有文件。
- ③ Close:关闭文件。
- ④ Open Workspace:打开工作区文件。
- ⑤ Save Workspace:保存工作区文件。
- ⑥ Close Workspace:关闭工作区文件。
- ⑦ Save:保存文件。
- ⑧ Save As:另存为。
- ⑨ Save All:保存打开的所有文件。
- ⑩ Page Setup: 页面设计。
- ⑪ Print:打印文件。
- ⑫ Recent Files:打开最近打开过的文件。

⑬ Recent Workspaces: 打开最近打开过的工作区文件。

⑭ Exit: 退出系统。

“Build”菜单用来编译、链接、调试和运行程序,如图 1.6 所示。各子菜单的中文意思如下。

- ① Compile: 编译程序代码。
- ② Build: 编译代码并链接工程。
- ③ Rebuild All: 重新编译并链接程序。
- ④ Start Debug: 进入调试状态。
- ⑤ Debugger Remote Connection: 远程调试设置。

VC++ 菜单还有很多,有的与 Microsoft Office 软件的菜单功能相似,有的是初学者暂时不必了解的内容,故在此不多介绍。

(2) 工具栏。

一般来说工具栏是菜单的快捷方式,所以工具栏中的工具一般都有相应的菜单。现将主要的工具介绍如下。

“Standard”工具栏用来建立项目工作区及项目,如图 1.7 所示。将鼠标指针停在其中一个图标上,就能出现关于该图标功能的说明文字。下面从左到右依次介绍如下。

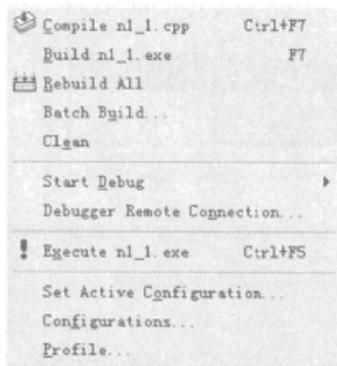


图 1.6 “Build”菜单



图 1.7 “Standard”工具栏

- ① New Text File: 创建新的文本文件。
- ② Open: 打开文件。
- ③ Save: 保存文档。
- ④ Save All: 保存所有打开的文档。
- ⑤ Cut: 剪切选定的内容。
- ⑥ Copy: 复制选定的内容。
- ⑦ Paste: 粘贴选定的内容。
- ⑧ Undo: 取消上一步操作。
- ⑨ Redo: 重复上一步操作。
- ⑩ Workspace: 显示/隐藏工作区窗口。
- ⑪ Output: 显示/隐藏输出窗口。
- ⑫ Windows list: 窗口管理。
- ⑬ Find in Files: 在多个文件中搜索。
- ⑭ Find: 查找字符串。
- ⑮ Search: 搜索联机文档。

“Build MiniBar”工具栏用来编译代码、链接目标文件和调试运行程序,如图 1.8 所示。

下面从左到右依次介绍如下。



图 1.8 “Build MiniBar”工具栏

- ① Compile:编译文件。
- ② Build:建立项目。
- ③ Stop Build:停止建立。
- ④ Execute Program:运行程序。
- ⑤ Go:启动或继续程序的执行。
- ⑥ Insert/Remove Breakpoint:插入或删除断点。

如果要了解 VC++ 的其他菜单功能,请参考有关书籍。

3. 程序开发步骤

用 VC++ 开发 C 语言程序有两种方法,下面先介绍单个文件开发的方法,对于工程文件的开发方法将放到后面介绍。

打开 VC++ 软件之后,在菜单栏中点击“File”菜单,在弹出的下拉菜单中点击“New”菜单,系统弹出“New”对话框,在对话框中选择“File”选项卡,再选择“C++ Source File”,在“File”下面填上源文件名,注意一定要加上扩展名.c,在“Location”下面选择源文件的存放目录,如图 1.9 所示。(加上.c 才是 C 程序,否则系统自动加上.cpp 扩展名,cpp 是 C++ 源程序的扩展名。)

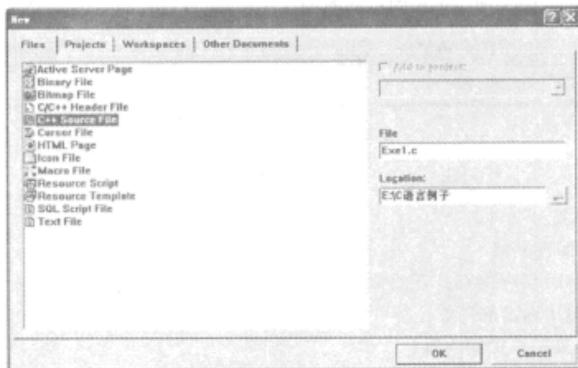


图 1.9 新建 C 语言源程序对话框

上述工作做完后,单击“OK”按钮,回到编辑窗口,此时编辑窗口为白色,并且有输入光标点,可进行编辑、编译、链接、运行等操作。

1.3 扩展知识与理论

在程序的编写过程中,一般要考虑两个方面的问题:一是数据结构,即程序中所要处理的数据对象以及它们之间的相互关系;另一个就是算法,即指对这些数据对象进行处理

时的求解方法。瑞士著名计算机科学家沃思(Nikiklaus Wirth)将其归纳为:程序=数据结构+算法。数据是程序处理的对象,而算法是程序的灵魂。

1.3.1 算法的概念

算法是指对完成一个任务准确而完整的描述。也就是说,给定初始状态或输入数据,经过计算机程序的有限次运算,能够得出所要求或期望的结果。

算法是解题的步骤,可以把算法定义成求解某一确定问题的任意一种特殊的方法。在计算机科学中,算法要用计算机算法语言来描述,算法代表用计算机解决问题的精确、有效的方法。求解一个给定的可计算或可解的问题,不同的人可以设计出不同的算法。

算法一般要满足下列五点。

- (1) 有穷性:一个算法必须保证在有限次运算后能够结束。
- (2) 确切性:算法的每一步骤必须有确切的定义,不能产生歧义或二义性。
- (3) 输入:一个算法有零个或多个输入,以确定运算对象的初始情况。所谓零个输入是指算法本身确定了初始条件。
- (4) 输出:一个算法有一个或多个输出,以反映对输入数据加工后的结果。没有输出的算法是毫无意义的算法。

(5) 可行性:算法原则上能够精确地运行,而且人们用笔和纸做有限次运算后即可完成。

好的算法的一般条件:① 正确性;② 可读性;③ 高效率与低存储量。

1.3.2 算法的描述方法

在设计一个较大任务的算法时,通常采用“自顶向下、逐步细化”的模块化程序设计方法,即将较大的任务按照一定的原则拆分为一个个较小的任务,一个程序模块完成一个小任务,这些小模块既相互关联又相互独立,而且容易理解。“班级学生成绩管理系统”设计就遵循了这一原则。

算法并不神秘,人们的生产活动和日常生活离不开算法,都在自觉或不自觉地使用算法。例如,人们到商店购买物品,会首先确定购买哪些物品,准备好所需的钱,然后确定到哪些商场选购,怎样去商场、行走的路线如何;若物品的质量好如何处理,对物品不满意又怎样处理,购买物品后做什么,等等。以上购物的算法是用自然语言描述的,也可以用其他描述方法描述该算法。

描述算法的一般方法如下:

- (1) 自然语言;
- (2) 图形,如流程图、N-S图,图的描述与算法语言的描述对应;
- (3) 算法语言,即计算机语言、程序设计语言、伪代码。

用自然语言来描述算法的好处是通俗易懂,缺点是过于冗长,易产生二义性。它一般适用于简单算法的描述。

用图形来描述算法是一种常用的方法,图形描述简洁明了,由于与算法语言描述对应,因此便于移植。常用的图形描述有流程图描述与N-S图描述。

流程图常用符号如图1.10所示。

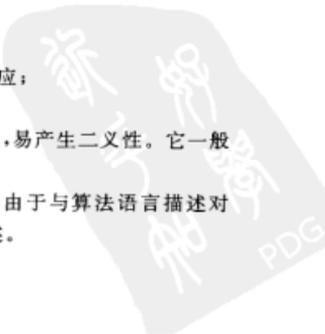




图 1.10 流程图常用符号

三种基本结构的 N-S 图如图 1.11 所示。

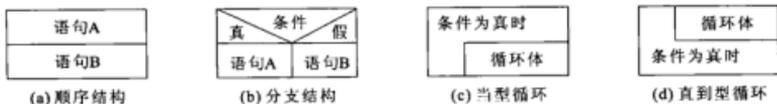


图 1.11 三种基本结构的 N-S 图

N-S 图的最大特点是将一个模块所完成的功能全部放在一个矩形框中,这更符合模块化设计的思想,同时 N-S 图更加形象直观,一目了然。不足之处是修改比较困难。

算法设计好后,必须通过计算机语言编写程序代码才能实现其功能,所以用计算机语言来实现算法是算法设计的最终目的。

1.4 深入训练

- (1) 模仿例 1.1 的方法,编写一个 C 语言程序,要求显示如下结果。

```
*****
How are you!
*****
```

- (2) 模仿例 1.1 的方法,编写一个 C 语言程序,显示下列结果。

```
| * * * * 学生成绩管理系统 * * * * |
| .....|
|           请选择菜单序号(0~6)       |
| .....|
|           1---打开文件               |
|           2---保存文件               |
|           3---编辑数据               |
|           4---显示数据               |
|           5---数据计算               |
|           6---程序说明               |
|           0---退出系统               |
| .....|
```

- (3) 模仿例 1.2 的方法,编写一个 C 语言程序,计算半径 R 等于 5 和 10 的圆面积。已知圆面积公式为: $S = R * R * 3.14159$ 。

- (4) 模仿例 1.3 的方法,编写一个 C 语言程序,计算圆柱体的体积。其中,底面半径

等于 2.5, 高等于 5。

(5) 编写一个 C 语言程序, 要求显示如下结果。

```

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

```

(6) 华氏温度与摄氏温度转换关系公式为: $C = 5/9.0 * (F - 32)$, 编写一个 C 语言程序, 将 100 华氏温度转换成摄氏温度, 并在屏幕上显示。

(7) 利用流程图和 N-S 图, 画出例 1.1 和例 1.2 的工作流程图。

习 题 1

1.1 填空题

- (1) C 语言规定, 一个程序必须有一个主函数, 其函数名为_____。
- (2) 一般而言, 一个 C 语言程序的执行是从_____开始, 到_____结束。
- (3) 一个 C 语言程序是由_____组成的。
- (4) 函数由_____和_____两部分组成。函数体又包括_____和_____两部分。
- (5) 开发 C 语言程序的步骤可以分成四步, 即_____、_____、_____、_____。
- (6) 用 Visual C++ 开发程序时, 编译按_____快捷键, 运行按_____快捷键。
- (7) 用 Visual C++ 开发 C 语言程序时, 一定不要忘记在文件前面加上_____头文件。
- (8) 在开发 C 语言程序时一般会出现两种错误, _____和_____错误, 语法错误一般是由_____引起的。怎样避免_____。
- (9) 用 Visual C++ 开发 C 语言程序有两种注释方法, 一种是_____, 另一种是_____, 能进行多行注释的是_____, 只能进行单行注释的是_____。
- (10) C 语言规定, 源程序的扩展名是_____, 目标文件的扩展名是_____, 可执行文件的扩展名是_____。

1.2 判断题(判断下列叙述的正确性, 正确的请打“√”, 错误的请打“×”)

- (1) 一个 C 语言的源程序由一系列函数组成。 ()
- (2) C 语言的任何一个源程序中必须有一个主函数。 ()
- (3) Visual C++ 6.0 不可以开发 C 语言程序。 ()
- (4) Visual C++ 是运行在 Windows 操作系统上的 32 位 C 语言程序开发工具。 ()
- (5) 算法就是一种解决问题的方法。 ()

单元2 项目数据设计与数据运算

能力目标

- 能够定义各种简单类型的常量与变量。
- 掌握数值常量、字符常量和符号常量的定义方法。
- 初步学会利用 C 语言常用的 7 种运算符和它们的表达式解决现实中的相关问题。
- 能进行不同类型数据之间的混合运算。
- 能设计“班级学生成绩管理系统”中所涉及到的简单类型常量和变量。

知识目标

- 理解 C 语言中的数据与现实中的数据区别与联系,理解数据类型的概念。
- 理解定义变量的格式、方法和关键字。
- 理解变量在内存中如何开辟存储空间,理解变量名与该空间的关系。
- 理解字符数据在内存中的存储方式。
- 理解 C 语言各种运算的运算规则,由运算符和相关数据组成表达式的方法。

学习提示

掌握数据类型以及不同类型的数据在内存中占用空间的不同,掌握变量概念、变量定义、运算符的运算规则以及不同类型之间的混合运算,这些是学习 C 语言的重要内容。学习本单元后读者应能正确设计“班级学生成绩管理系统”中有关的变量和常量。

2.1 任务 2:“班级学生成绩管理系统”中相关数据设计

“班级学生成绩管理系统”中的学生信息主要包括:学号、姓名、性别、年龄、三门功课成绩、总成绩和平均成绩,再加上一些与计算全班成绩有关的最高成绩、最低成绩等。这些信息在程序运行过程中是可能改变的,这里给出部分根据学生信息设计的变量,在定义变量时最好能做到“见名识意”。另外,一个班的学生总人数控制为 40 人,这个数据在程序的运行过程中是不变的。

```

int stunum;           //整数类型的学号
char stusex;         //字符类型的性别
int stuage;          //整数类型的年龄
float score1;         //单精度类型的成绩 1
float score2;         //单精度类型的成绩 2
float score3;         //单精度类型的成绩 3
float avscore;        //单精度类型的平均成绩
float maxscore;       //单精度类型的最高分
float minscore;       //单精度类型的最低分

```

学习时,要注意不同的数据使用了不同的类型,不同的类型在内存空间中占用的字节多少是不同的,还要注意不同类型数据的运算方法与运算结果也是不同的。还有一些变量(如姓名)比这些简单变量要复杂,我们将在后续内容中加以讲解。

单元能力训练任务分别如下。

任务 A:正确使用简单数据类型关键字定义各种类型变量并设计各种常量。

任务 B:运用赋值、算术、关系、逻辑、逗号、sizeof 等运算符构成相应表达式。

任务 C:设计“班级学生成绩管理系统”中所涉及的变量和常量。

任务 D:模仿任务 2 设计“学生通讯录”中所涉及的变量和常量。

2.2 必备知识与理论

2.2.1 数据类型概述

数据是程序处理的对象,计算机中处理的数据与自然界的的数据有什么不同呢?自然界中的数据只有数值大小之分,表示范围可以从负无穷到正无穷。而计算机要处理的数据是放在计算机内存中的,因此它除了有数值大小之分之外,还有占用存储空间(占几个字节)大小之分,这种存储空间大小,决定了实际存储数据的大小,也就是说计算机内存中不能保存无穷大或无穷小的数据。另外,计算机中的数据,无论是什么类型的数据,都将转变成二进制数据存放,而不是我们所熟悉的十进制或字符。

计算机中的数据不像自然界中的数据,不能表示任意大小。因为计算机中的数据大小受存储空间的限制,这种限制是由数据类型来决定的。在 C 语言中不同的数据都被规定在不同的数据类型中,因此 C 语言是一种强类型的语言,即数据必有类型。我们可以通过数据类型得到数据在内存中占用了多少内存空间,也可以通过 sizeof 表达式来计算数据占用空间的大小。

C 语言的数据类型分为简单数据类型和复杂数据类型两种。简单数据类型是由系统自动规定数据存储空间的大小,它包括整数类型(整型)、实数类型(实型)、字符类型(字符型)和空类型。而复杂数据类型是由用户按照一定规则来决定数据占用空间的大小,它包括数组类型、结构体类型、共用体类型和枚举类型,因此后者也可以称为用户自定义类型。

C 语言数据类型如图 2.1 所示。



图 2.1 C 语言数据类型

2.2.2 常量与变量

1. 常量

常量是指在程序运行过程中,其值始终保持不变的量。常量可分为不同的类型,如整型常量 0、-3;实型常量 4.6、-1.23;字符常量 'd';字符串常量 "a"。

常量从形式上又可分为字面(直接)常量和符号常量两种。从字面上就可以判别的常量称为字面常量,用一个标识符代表一个常量的称为符号常量。

【例 2.1】 已知圆的半径,求圆的面积与周长。圆周率定义为符号常量。

```

1 #include <stdio.h>
2 #define PI 3.14159f /*将圆周率定义为符号常量*/
3 main()
4 {
5     float r,s,l;
6     r=1.5;
7     s=r*r*PI;
8     l=2*r*PI;
9     printf("s=%f\n",s);
10    printf("l=%f\n",l);
11 }
  
```

程序运行结果:

```

s=7.068578
l=9.424770
  
```



例 2.1 中用 #define 命令定义 PI 代表常量 3.14159,以后在程序中凡出现的 PI 均代表 3.14159。因此,PI 是一个符号常量,它不同于变量,它的值在它的作用域内不能改变,也不能再被赋值。习惯上,符号常量名用大写,变量名用小写,以示区别。注意:这也仅仅是习惯上的用法,而不是规定。

2. 变量

变量是指在程序运行中,其值可以改变的量。一个变量应该有一个名字,在内存中占据一定的存储空间。

变量名是用标识符来表示的。C 语言规定,标识符只能由字母、数字和下划线三种字符组成,且第一个字符必须为字母或下划线。另外,C 语言对大小写敏感,因此要注意变量名字母的大小写,如 sum 和 SUM 系统认为是两个不同的变量名。

C 语言规定,对所用到的变量要做强制定义,即“先定义,后使用”。如果不定义就使用,系统会提示“Undefined symbol 'xxxx'”,其意为符号“xxxx”未定义。定义变量名时还要尽量做到“见名知意”。

初学变量时要特别注意区分变量名、变量值、变量地址,如图 2.2 所示。

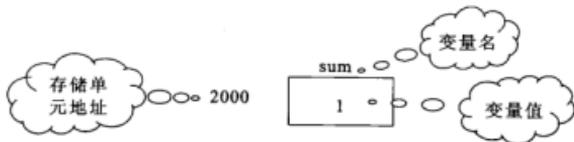


图 2.2 变量名、变量值、变量地址的关系图

2.2.3 简单数据类型

1. 整型数据

1) 整型常量的表示方法

整型常量即整型常数,可用如下三种形式表示。

- (1) 十进制整型常量,有正、负数之分。如 123、-456、0。
- (2) 八进制整型常量,以 0 开头,有正、负数之分。如 0123、-034、075。
- (3) 十六进制整型常量。以 0x 或 0X 开头,有正、负数之分。如 0x123、0XDC、-0x34e。

2) 整型变量

(1) 整型数据在内存中的存放形式。

数据在内存中以二进制形式存放。如十进制数 5 的二进制形式为 101,由于 VC++ 是 32 位的编译系统,因此系统规定每个整型变量在内存中占 4 个字节(而 16 位编译系统如 TC 占 2 个字节)。整数 5 在内存中存储示意如图 2.3 所示。



图 2.3 整数 5 在内存中存储示意图

(2) 整型变量的分类与数值范围如表 2.1 所示。

表 2.1 整型变量的分类与数值范围

类别	类型标识符(关键字)	类型名称	存储空间	值域
短整型	short short int signed short signed short int	有符号短整型	2 字节	-32768~32767 间的整数
	unsigned short unsigned short int	无符号短整形	2 字节	0~65535 间的整数
整型	int signed signed int	有符号整型	4 字节	-2147483648~2147483647 间的整数
	unsigned unsigned int	无符号整形	4 字节	0~4294967295 间的整数
长整型	long long int signed long signed long int	有符号长整型	4 字节	-2147483648~2147483647 间的整数
	unsigned long unsigned long int	无符号长整型	4 字节	0~4294967295 间的整数

3) 整型变量的定义与初始化

在前面已经说过,变量必须先定义然后才能使用,对变量的定义一般是放在函数开头的声明部分。

定义格式:

类型标识符 变量名[, 变量名, ……];

例如: int x;

int i, j;

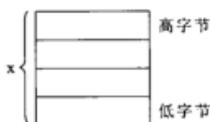


图 2.4 变量 x 在内存中开辟 4 个字节的存储空间

注意:定义变量后,程序链接时由系统为该变量在内存中开辟(分配)存储空间。如定义整型变量 x,程序链接时在内存中开辟 4 个字节的存储区,并将该存储区命名为 x,如图 2.4 所示。

定义变量后可以给它赋值, $x=5$ 。赋值后的内存情况如图 2.3 所示。如果是在定义变量的同时给变量赋值,这种赋值方法称为变量初始化。

例如: int x=5;

在实际编程工作中,如何来选择整数的类型呢?下面举一个给学号设置类型的例子来说明这个问题。

假定:学号由年号(4位)、系列号(2位)、班级号(2位)、序号(2位),共 10 位组成。我们知道短整型无论是有符号还是无符号都只有 5 位。因此都不能容纳下 10 位数,必须选择整型或长整型,因为它们的值域范围,有符号是 $-2147483648 \sim 2147483647$ 间的整数,无符号是 $0 \sim 4294967295$ 间的整数,都能容纳下 10 位数的学号。所以,学号类型应选择整

型或长整型,定义为 int stunum 或 long stunum 均可。

2. 实型数据

1) 实型常量的表示方法

(1) 实型数据又称为浮点数,有以下两种常量表示形式。

① 十进制小数形式。它由数字和小数点组成,如 0.123、.123、123.、123.0 等。

② 十进制指数形式。它由“尾数 e(或 E)指数”形式组成,如 123e3、123E3,但要注意 e 或 E 前必须有数字,e(E)后的指数必须为整数。一个实数有多种指数表示形式,但规范化的指数形式只有唯一一种,如图 2.5 所示。

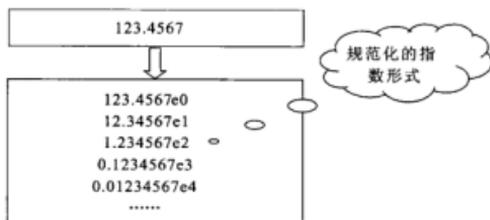


图 2.5 实型数据的指数表示形式

规范化的指数形式是指字母 e(或 E)之前的小数部分中,小数点左边应有一位,且只能有一位非零的数字。

(2) 实型常量的类型。VC++ 编译系统将实型常量作为双精度型来处理,如果一定要将实型常量表示为单精度型,可以在实型常量后加 f。如例 2.1 中第 2 行常量 3.14159 后加了 f,3.14159 就是单精度实型常量了。

2) 实型变量

(1) 实型数据在内存中的存放形式:一个实型数据在内存中占 4~8 个字节,实型数据是按指数形式存储的。

(2) 实型变量的分类。实型变量可分为单精度(float)、双精度(double)两类,实型变量的分类与数值范围如表 2.2 所示。

表 2.2 实型变量的分类与数值范围

类别	类型标识符	存储空间	值域	有效位
单精度	float	4 字节	$3.4 \times 10^{-37} \sim 3.4 \times 10^{38}$	6~7
	double			
双精度	double float	8 字节	$1.7 \times 10^{-307} \sim 1.7 \times 10^{308}$	15~16
	long float			

(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,因此提供的有效数字是有限的,在有效位以外的数字是不准确的,由此就会产生一些误差。

3) 实型变量的定义与初始化

定义格式:

类型标识符 变量名[, 变量名, ……];

例如: float a;
double b;

注意: 定义实型变量 a 后, 系统将在内存开辟 4 个字节的存储区, 并将该存储区命名为 a, 定义实型变量 b 后, 系统将在内存中开辟 8 个字节的存储区, 内存开辟空间如图 2.6 所示。

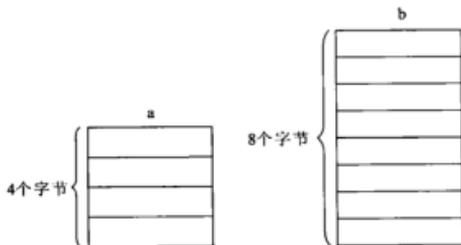


图 2.6 变量 a、b 在内存开辟空间示意图

定义变量后可以给它赋值: $a=1234.567$; 如果是在定义变量的同时给变量赋值, 这称为变量初始化。例如, $\text{float } a=1234.567$ 。

3. 字符型数据

1) 字符型常量

字符型常量分为直接显示字符常量和转义字符常量。

(1) 直接显示字符: 除单引号(')、双引号(")和反斜杠(\)之外的可显示字符, 用该字符直接表示, 但必须用单引号括起来。例如, 'a'、'x'等。

(2) 转义字符: 以反斜杠"\ "开头的字符称为“转义字符”。可以这样理解: 反斜杠后面的字符已不是原来的含义, 它的含义已经发生了变化, 所以称为“转义”。转义字符如表 2.3 所示。

表 2.3 转义字符

字符形式	含义	字符形式	含义
\n	LF(换行)	\a	BEL(鸣响)
\t	HT(横向跳格)	\\	反斜杠
\v	VT(纵向跳格)	\'	单引号
\b	BS(退格)	\"	双引号
\r	CR(回车)	\ddd	八进制数
\f	FF(换页)	\xhh	十六进制数

2) 字符型变量

对于字符型变量,一般情况下不必考虑有符号的情况,只需要考虑无符号的情况。它的类别与取值范围如表 2.4 所示。

表 2.4 字符型变量的分类与取值范围

类别	类型标识符	存储空间	值域
字符型	char	1 个字节	0~255

3) 字符变量的定义与初始化

定义格式:

类型标识符 变量名[, 变量名, ……];

例如:char ch;

char ch1, ch2;

定义字符变量 ch 后,系统将在内存开辟 1 个字节的存储区,并将该存储区命名为 ch,如图 2.7 所示。

字符变量是用来存储字符常量的。因此,定义字符变量后可以给它赋值:ch='A';如果是在定义变量的同时给变量赋值,则称为变量初始化。例如,char ch='A'。

想一想:这个单元中存放的是字符 A 吗?



图 2.7 变量 ch 在内存中开辟空间示意图

4) 字符型数据在内存中的存储形式及其使用方法

(1) 存储形式:将一个字符常量存放到字符变量中,并不是把该字符本身存放到内存单元中,而是将该字符相应的 ASCII 代码存放到该存储单元中。

例如,char ch='A';字符常量 A 在内存中存储如图 2.8 所示。



图 2.8 字符常量 A 在内存中存储示意图

(2) 使用方法:字符数据是以 ASCII 码存储的,这在本质上和整型数据的存储是类似的。这样,字符型数据和整型数据之间的转换就变得非常方便了。

一个字符数据既可以以字符形式输出,也可以以整数形式输出。以字符形式输出时,需先将存储单元中的 ASCII 码转换成相应的字符,然后输出;以整数形式输出时,直接将 ASCII 码作为整数输出。

也可以对字符数据进行算术运算,此时相当于对它们的 ASCII 码进行算术运算,只将其中字符占用的一个字节转化为 4 个字节,然后参加运算。

请看下面的例子。

【例 2.2】 字符与整数转换。

```
1 #include <stdio.h>
2 void main()
3 {
4     char c1,c2;
5     c1='A';
6     c2=98;
7     printf("%c %c\n",c1,c2);/*使用了%c的输出格式,以字符形式输出*/
8     printf("%d %d\n",c1,c2);/*使用了%d的输出格式,以整数形式输出*/
9 }
```

程序运行结果:

```
A b
65 98
```

【例 2.3】 将小写字母转换成大写字母的函数。

```
1 #include <stdio.h>
2 int UPPER(char c)
3 {
4     int ch;
5     if(c>='a' && c<='z')
6         ch=c-32;
7     return ch;
8 }
9
10 void main()
11 {
12     char ch;
13     scanf("%d",&ch); //输入一个整型数据
14     printf("%c\n",UPPER(ch));
15 }
```

程序运行结果:

```
97<回车>
A
```

从以上两个实例可以得出,字符型数据与整型数据由于它的存储本质是相同的,所以可以很容易实现它们的相互转换和混合运算。

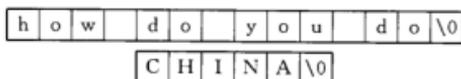
5) 字符串常量

字符串常量是由一对双引号括起来的字符序列。如“how do you do”、“CHINA”、“a”、“\$ 123.45”都是字符串常量。注意字符常量与字符串常量之间的区别。

字符串中的每个字符占一个字节,字符串有结束标记‘\0’,即每个字符串都在末尾自



动加上一个结束标志'\0'。例如：



字符 A 与字符串 A 在存储结构上有区别吗？回答是肯定的，如图 2.9 所示。



图 2.9 字符 A 与字符串 A 占用内存区别示意图

由图 2.9 可以看出字符 A 在内存中占用一个字符，而字符串 A 除了存放字符本身之外，还将存储一个字符串结构标志'\0'，因此，字符串占用的空间应当是字符串中字符个数加 1。另外，C 语言没有专门的字符串变量，如果想将一个字符串存放在变量中，必须使用字符数组。关于数组将在后面介绍。

2.2.4 数据运算符及其表达式

C 语言运算符范围很广，除控制语句和输入/输出语句之外，几乎所有的基本操作都作为运算符处理。C 语言的运算符有以下几类。

- ① 赋值运算符：= 及其复合赋值运算符。
- ② 算术运算符：+、-、*、/、%、++、--。
- ③ 关系运算符：>、<、==、<=、>=、!=。
- ④ 逻辑运算符：!、&&、||。
- ⑤ 逗号运算符：，。
- ⑥ 求字节数运算符：sizeof。
- ⑦ 强制类型转换运算符：(类型)。
- ⑧ 位运算符：<<、>>、~、|、-、&。
- ⑨ 条件运算符：?:。
- ⑩ 下标运算符：[]。
- ⑪ 指针地址运算符：* 和 &。
- ⑫ 分量运算符：.、->。

这里主要介绍赋值运算符、算术运算符、关系运算符、逻辑运算符、逗号运算符和求字节数运算符。并将位运算作为扩展知识与理论，其他的运算符放到后面介绍。

1. 赋值运算符及其表达式

1) 赋值运算符

赋值符号“=”就是赋值运算符，它的作用是将一个数据赋给一个变量。

例如， $a=3$ ，其作用是把常量 3 赋给变量 a。

也可以将一个表达式的值赋给一个变量。

例如， $x=y+5$ ，其作用是将表达式 $y+5$ 的和赋给变量 y。

赋值运算符的优先级比较低，结合性是自右至左。

2) 类型转换

进行赋值运算时,一般要求赋值运算符两侧的数据类型一致或兼容。

如果赋值运算符两侧的类型不一致,但都是数值型或字符型,在赋值时可以进行类型转换。

(1) 将实型数据(包括单、双精度)赋给整型变量时,舍弃实数的小数部分。

(2) 将整型数据赋给单、双精度变量时,数值不变,但以浮点数值形式存储到变量中。

由于 C 语言使用灵活,在不同类型数据之间赋值时,常常会出现意想不到的结果,而编译系统并不提示出错,全靠程序员的经验来找出问题。这就要求编程人员对出现的原因有所了解,以便迅速排除故障。对学习来说最好的办法是多读书、多实践,从上机实践中获得经验。

3) 复合赋值运算符

在赋值符“=”前加上其他运算符,可以构成复合的运算符。如在“=”前加上“+”运算符就成了复合运算符“+=”。例如,

$a+=3$ 等价于 $a=a+3$

$x*=y+8$ 等价于 $x=x*(y+8)$ //括号是必要的,不能省

$x\%=3$ 等价于 $x=x\%3$

复合赋值运算符通常格式:

变量? =表达式

等价于:

变量=变量?(表达式) //其中?表示其他运算符

凡是二元(二目)运算符,都可以与赋值符一起组成复合赋值符。C 语言规定可以使用 10 种复合赋值运算符,如表 2.5 所示。

表 2.5 复合赋值运算符

符 号	名 称	符 号	名 称
+=	加赋值	<<=	左移赋值
-=	减赋值	>=	右移赋值
*=	乘赋值	&=	按位与赋值
/=	除赋值	=	按位或赋值
%=	求余赋值	^=	按位异或赋值

4) 赋值表达式

由赋值运算符将一个变量和一个表达式连接起来的符合 C 语言运算规则的式子称为赋值表达式。

其一般格式:

< 变量> < 赋值运算符> < 表达式>

例如, $a = 7 \% 2$ 。

对赋值表达式的求解过程是:先将赋值运算符右侧的“表达式”计算出来后,再将其值赋给左侧的变量,即从右向左计算。赋值表达式的值就是被赋值的变量的值。

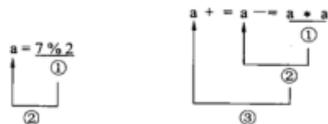


图 2.10 赋值表达式求解过程示意图

赋值表达式求解过程如图 2.10 所示。

赋值表达式中的“表达式”,又可以是一个赋值表达式。例如,

$a = (b = 5)$ (先将 5 赋给 b,然后再将表达式 $b = 5$ 的值赋给 a)
 $a = b = c = 5$ (赋值表达式的值为 5, a、b、c 的值均为 5)
 $a = 5 + (c = 6)$ (赋值表达式的值为 11, a 的值为 11, c 的值为 6)
 $a = (b = 4) + (c = 6)$ (赋值表达式的值为 10, a 的值为 10, b 等于 4, c 等于 6)

由此可以看出:赋值表达式有两个值,一个是表达式的值,一个是变量的值。

赋值表达式也可以包含复合的赋值运算符,例如,

$a += a - = a * a$

也是一个赋值表达式,也遵循“自右至左”计算的规则,如图 2.10 所示。

想一想:如果 a 的初值为 8,计算后表达式的值为多少? a 的值又为多少? 将赋值表达式作为表达式的一种,使赋值操作不仅可以出现在赋值语句中,而且可以以表达式形式出现在其他语句(如输出语句、循环语句等)中,例如,

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

上面语句实际上完成两件事,一方面将 b 值赋给 a,另一方面将表达式的值输出。如果 b 的初值为 4,首先将 4 赋给 a,表达式的值也为 4,输出 4。所以,这条语句完成了赋值和输出双重功能。这也是 C 语言的优越性之一。

要注意的是,赋值符左侧必须为一变量(该变量又可以称为左值),不能是常量和表达式,因为变量在内存中开辟空间,可以用来存储给定的值,而常量的值是不能改变的,内存也不为表达式开辟存储空间。

2. 算术运算符

1) 算术运算符

- (1) +: 加法运算符,或正值运算符。如 $3 + 5$ 、 $+3$ 等。
- (2) -: 减法运算符,或负值运算符。如 $5 - 2$ 、 -3 等。
- (3) *: 乘法运算符。如 $3 * 5$ 等。
- (4) /: 除法运算符。如 $5/3$ 、 $5.0/3$ 等,“/”两侧均为整型数据,商为整型数据,有一侧为实型或两侧均为实型数据,商为实型数据。
- (5) %: 模运算符或求余运算符。“%”两侧只能为整型数据,如 $7 \% 4$ 的值为 3。
- (6) ++: 自增运算符。

(7) --:自减运算符。

2) 算术运算符的优先级与结合性

用算术运算符和括号将运算对象连接起来的符合C语言语法规则的式子,称为算术表达式。运算对象包括常量、变量、函数等。

C语言规定了运算符的优先级和结合性(见附录IV)。算术运算符中自增、自减运算符的优先级比较高,高于其他算术运算符的优先级,自增、自减运算符的结合性是“自右至左”,而其他算术运算符的结合性是“自左至右”。在表达式求值时,先按运算符的优先级由高向低顺序执行,如先乘除后加减。当运算符的优先级相同时,运算方向由“结合性”决定。

3) 自增、自减运算符

自增、自减运算符的作用是使变量的值增1或减1。它实际上是形如 $x=x+1$ 或 $x=x-1$ 运算的简洁的书写方法。

自增、自减运算符又分为两种形式。

(1) 前自增、前自减运算。运算符写在变量前面,称为前自增、前自减。如 $++k$ 、 $--k$ 。

运算规则:变量先加1或减1,然后再使用变量。

例如, $x=5$; $y=++x$;

问 x 和 y 的值各是多少?

$y=++x$ 语句相当于先进行 $x=x+1$ 运算,再进行 $y=x$ 运算,显然 x 和 y 都等于6。

(2) 后自增、后自减运算。运算符写在变量后面,称为后自增、后自减。如 $k++$ 、 $k--$ 。

运算规则:变量先使用,然后变量再加1或减1。

例如, $x=5$; $y=x++$;

问 x 和 y 的值是多少?

$y=x++$ 语句相当于先进行 $y=x$ 运算,再进行 $x=x+1$ 运算,显然 $y=5$, $x=6$ 。

自增运算符($++$)和自减运算符($--$),只能用于变量,而不能用于常量或表达式,如 $4++$ 或 $(a+b)++$ 都是不合法的,因为常量的值是不能改变的,表达式不能存放自增或自减后的值。

$++$ 和 $--$ 的结合性是“自右至左”,算术运算符的结合性是“自左至右”,如果有 $-i++$,怎么结合呢? i 的左边是一个负值运算符,是算术运算符,结合性为自左至右; i 的右边是自增运算符,结合性为自右至左。如果是左结合,就是 $(-i)++$ 。而 $(-i)++$ 是非法的,因为表达式不能进行自增运算。所以应该是 $-(i++)$ 。

请问: $\text{printf}("%d, %d", -i++, i)$;的值为多少,假定 i 的初值为5。

程序执行结果为:-5,6。

自增自减运算符常用于循环语句中,使循环变量自动加(减)1,也可用于指针变量,使指针向下(上)移动一个单元。

自增、自减运算对象的类型可以是整型、实型和字符型。

3. 关系运算符及其表达式

所谓“关系运算”实际上就是“比较运算”，即将两个数据进行比较，判定两个数据是否符合给定的关系。

例如，“ $a > b$ ”中的“ $>$ ”表示一个大于关系运算。如果 a 的值是 5， b 的值是 3，则大于关系成立，其结果为“真”；如果 a 的值是 2， b 的值是 3，则大于关系不成立，其结果为“假”。

1) 关系运算符

C 语言提供 6 种关系运算符，如表 2.6 所示。

表 2.6 关系运算符

运算符	说明	运算符	说明	运算符	说明
$==$	等于	$>$	大于	$>=$	大于等于
$!=$	不等于	$<$	小于	$<=$	小于等于

注意：在 C 语言中，“等于”关系运算符是双等号“ $==$ ”，而不是单等号“ $=$ ”（赋值运算符）。

2) 关系运算符的优先级

(1) 在关系运算符中，大于、小于、大于等于、小于等于的优先级别相同，等于、不等于的优先级别相同，前者的优先级别高于后者。

(2) 与其他种类运算符的优先级关系：关系运算符的优先级别低于算术运算符，但高于赋值运算符和逗号运算符。

3) 关系表达式

所谓关系表达式是指，用关系运算符将两个表达式连接起来的符合 C 语言规则的式子。关系表达式是进行关系运算的式子。

例如，下面的关系表达式都是合法的：

$a > b$, $a + b > c - d$, $(a = 3) <= (b = 5)$, $'a' > 'b'$, $(a > b) == (b > c)$

关系运算符都是双目运算符。

4) 关系表达式的值

由于 C 语言没有逻辑类型，C 语言的逻辑值用整型数据 1（非 0 值）表示“逻辑真”（true），用整型数据 0（0 值）表示“逻辑假”（false）。

例如，假设 $num1 = 3$, $num2 = 4$, $num3 = 5$ ，则：

$num1 > num2$ 的值为 0；

$(num1 > num2) ! = num3$ 的值为 1；

$num1 < num2 < num3$ 的值为 1。

如果任意改变 num1 或 num2 的值,会影响整个表达式的值吗? 为什么?
 $(num1 < num2) + num3$ 的值为 6, 因为 $num1 < num2$ 的值为 1, 则 $1 + 5 = 6$ 。

4. 逻辑运算符及其表达式

关系表达式只能描述单一条件,如“ $x > 0$ ”。如果需要描述“ $x > 0$ ”同时“ $x < 10$ ”,就要借助于逻辑表达式了。

1) 逻辑运算符及其运算规则

(1) 逻辑运算符如表 2.7 所示。

表 2.7 逻辑运算符

运算符	说 明
&&	逻辑与(相当于“同时”)
	逻辑或(相当于“或者”)
!	逻辑非(相当于“否定”)

(2) 运算规则。

① &&: 当且仅当两个运算量的值都为“真”时,运算结果为“真”,否则为“假”。

② ||: 当且仅当两个运算量的值都为“假”时,运算结果为“假”,否则为“真”。

③ !: 当运算量的值为“真”时,运算结果为“假”;当运算量的值为“假”时,运算结果为“真”。

例如,假定 $x=5$,则 $(x > 0) \&\& (x < 10)$ 的值为“真”, $(x < -1) || (x > 5)$ 的值为“假”。

又如,年满 18 岁的男生,可表示为年满 18 岁 && 男生;妇女和儿童,可表示为妇女 || 儿童;等等。

2) 逻辑运算符的优先级

(1) 逻辑非的优先级最高,逻辑与次之,逻辑或最低。

! (非) \rightarrow && (与) \rightarrow || (或)

(2) 与其他种类运算符的优先关系。

! \rightarrow 算术运算符 \rightarrow 关系运算符 \rightarrow && \rightarrow || \rightarrow 赋值运算符 \rightarrow 逗号运算符

3) 逻辑表达式

所谓逻辑表达式,是指用逻辑运算符将一个或多个表达式连接起来,符合 C 语言规则的逻辑运算式子。在 C 语言中,用逻辑表达式表示多个条件的组合。

例如, $(year \% 4 == 0) \&\& (year \% 100 != 0) || (year \% 400 == 0)$ 就是一个判断某年份是否是闰年的逻辑表达式。

4) 逻辑表达式的值

逻辑表达式的值也是一个逻辑值(非“真”即“假”)。

C语言用整数1表示“逻辑真”、用整数0表示“逻辑假”。但在判断一个数据的“真”或“假”时,却以0和非0为根据,如果为0,则判定为“逻辑假”;如果为非0,则判定为“逻辑真”。

5) 说明

(1) 逻辑运算符两侧的操作数,除可以是0和非0的整数外,也可以是其他任何类型的数据,如实型、字符型等,但这些值都要根据规则看成是逻辑值。

(2) 在计算逻辑表达式时,只有在必须执行下一个表达式才能求解时,才求解该表达式(即并不是所有的表达式都被求解)。

① 对于逻辑与运算,如果第一个操作数被判定为“假”,系统不再判定或求解第二操作数。

② 对于逻辑或运算,如果第一个操作数被判定为“真”,系统不再判定或求解第二操作数。

5. 逗号运算符及其表达式

C语言提供一种特殊的运算符——逗号运算符,它将两个表达式连接起来。逗号运算符的优先级是最低的,结合性是自左至右。

例如,“3+5,6+8”称为逗号表达式,又称为“顺序求值运算符”。逗号表达式的一般格式为:

表达式 1, 表达式 2

逗号表达式的求解过程是:先求解表达式1,再求解表达式2。整个逗号表达式的值是表达式2的值。

如上面的逗号表达式“3+5,6+8”的值为14。

再如:

$a=3*5, a*4$ (若a的初值为3,表达式的值为多少呢?)

此题有两种分析。

(1) 相当于增加了一个括号: $a=(3*5, a*4)$,括号中是一个逗号表达式,求值后表达式的值为12,a的值也为12。

(2) 如果括号这样加 $(a=3*5), (a*4)$,先计算 $(a=3*5)$ 的值为15,a的值为15,再计算 $(a*4)$ 的值为60,因此逗号表达式的值也为60。

两种运算哪一个对呢?这还是要从运算符的优先级考虑。我们知道赋值运算符的优先级高于逗号运算符的优先级,当计算了 $3*5=15$ 后,式子变成 $a=15, a*4$,15左右两边有两个运算符,一个是“,”号,另一个是“=”号,显然优先级高的先运算,由于“=”的优先级高于“,”,因此15先赋给a。由此可见正确的算法应当是第二种。

一个逗号表达式又可以与另一个逗号表达式组成一个新的逗号表达式,例如:

$(a=3*5, a*4), a+5$

先计算a的值为15(a的值被改变),再计算 $a*4$ 的值为60(a的值没有改变),最后计算 $a+5$ 的值为20。逗号表达式的值也为20。

逗号表达式的一般格式可扩展为：

表达式 1, 表达式 2, 表达式 3, …… 表达式 n

它的值为表达式 n 的值。

由于逗号表达式的优先级最低，所以有无括号对表达式的值有着直接的影响，例如：

`x=(a=3,6*3) …… ①`

`x=a=3,6*a …… ②`

式①是一个赋值表达式，将一个逗号表达式的值赋给 x，x 的值为 18。

式②是逗号表达式，它包括一个赋值表达式和一个算术表达式，表达式的值为 18，但 x 的值为 3。

其实，逗号表达式无非是把若干表达式“串联”起来。在多数情况下逗号表达式只是想分别计算各表达式的值，而并不一定要得到和使用整个逗号表达式的值，逗号表达式一般用在循环语句（如 for 语句）中。

注意：并不是任何地方出现的逗号都作为逗号运算符。如函数参数也是用逗号来间隔的，但它的意义就不同了，如：`printf("%d,%d,%d",a,b,c)`；

这里的逗号并不表示逗号运算符，它只是三个参数的分隔符，如果将上面的输出语句改为：`printf("%d,%d,%d",(a,b,c),b,c)`；

它也有三个参数，不过第一个参数(a,b,c)是一个逗号表达式，括号中的逗号是逗号运算符，其值为 c，后面的两个逗号是参数分隔符。

6. sizeof 运算符

sizeof 运算符用来获得一个数据或数据类型在内存中所占空间的字节数。

格式：

sizeof(类型标识符)或 sizeof(表达式)

例如：

```
sizeof(int)           //获得整型数据所占内存空间的字节数
sizeof(double)       //获得双精度数据所占字节数
```

如果表达式是单个的常量或变量，括号也可以不要。

【例 2.4】 编程获得当前使用的编译系统所分配的数据类型、常量、变量、表达式所占内存的字节数。

```
1 #include <stdio.h>
2 void main( )
3 {
4     double f=3;
5     printf("%d ",sizeof(int));
6     printf("%d ",sizeof(long));
7     printf("%d ",sizeof(float));
8     printf("%d ",sizeof(double));
```

```

9     printf("%d ", sizeof(char));
10    printf("%d ", sizeof(5));
11    printf("%d ", sizeof(f));
12    printf("%d\n", sizeof(f+5));
13 }

```

程序运行结果：

```
4 4 4 8 1 4 8 8
```

同一种数据类型在不同的编译系统中所占空间不一定相同。例如，在基于 16 位的编译系统中，int 型数据占用 2 个字节，但现在普遍使用的是基于 32 位的编译系统，int 型数据要占用 4 个字节。因此为了便于移植程序，最好用 sizeof 运算符计算数据类型。

2.2.5 不同数值型数据间的混合运算

前面已经说过字符型数据与整型数据间是可以进行混合运算的。整型数据与实型数据间能不能进行混合运算呢？

这个问题看来是一个简单问题，因为小学生们都知道它们是可以进行混合运算的。如 $5+3.5=8.5$ 。既然现实生活中有这种计算的需要，C 语言一定能够进行整型、实型、字符型数据间的混合运算。那么，问题为什么会提出来呢？这主要是由于整型、实型和字符型数据在内存中所占用的空间不同，占用不同空间的数据能进行混合运算吗？

例如，‘a’+10+65.1234 的运算实际上就变成了占用 1 个字节、4 个字节、8 个字节数据之间的运算了，如图 2.11 所示。

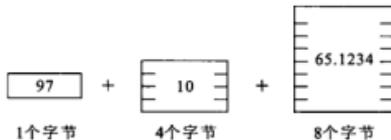


图 2.11 不同类型数据混合运算示意图

现在的问题是，它们按什么类型进行运算？运算后结果的类型又是什么？

设想一下，如果按占用字节少的类型来计算，这样计算的结果可能造成字节多的类型丢失数据，引起计算结果不准，误差很大，显然这是不可取的。因此，对于不同类型的数据进行混合运算，一般按占用字节数较多的类型进行计算。

1. 自动类型转换

在进行运算时，不同类型的数据要先转换成同一类型，然后进行运算，自动类型转换的规则如图 2.12 所示。

图 2.12 中，向左的箭头表示无条件自动转换；如字符型和 short 型无条件自动转换成整型。纵向的箭头表示运算对象为不同类型时自动转换的方向，如 int 型与 double 型数据进行运算，先将 int 型的数据转换成 double 型，然后在两个同类型数据间进行运算，结果为 double 型。总之，如果两个操作对象有一个是 float 型或 double 型，则另一个数据

要先转换为 double 型,运算结果为 double 型;如果参加运算的两个数据中最高级别为 long,则另一个数据先转换成 long 型,运算结果为 long。千万不要理解为什么类型的运算都转换成 double 型,也不要理解为转换是一级一级完成的。

如 int 转换成 double 不是 int→unsigned→long→double,而是直接转换成 int→double。

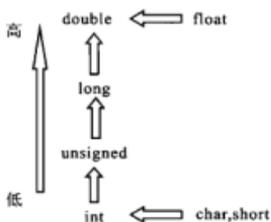


图 2.12 自动类型转换规则

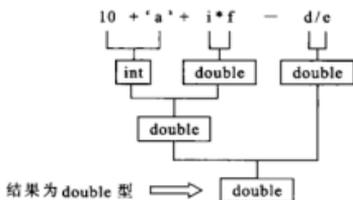


图 2.13 自动类型转换实例

转换实例:假定已指定 i 为整型变量, f 为 float 变量, d 为 double 型, e 为 long 型, 自动类型转换如图 2.13 所示。

注意:上述类型转换在 16 位 TC 2.0 编译系统中由系统无条件完成,而在 32 位的 VC++ 6.0 系统中, float 型不能无条件转换成 double 型。

2. 强制类型转换

如果数据由占用内存空间大的向占用内存空间小的方向转换,如实型转换成整型,整型转换成字符型,系统就不能自动完成转换过程,如果一定要转换,就只能利用强制类型转换运算符将变量或表达式转换成所需类型。

其一般格式为:

(类型标识符)(表达式)

如 $(int)(x+y)$,是将 $x+y$ 的值转换成整型。如果写成 $(int)x+y$,表示只将 x 的类型强制转换成整型,而不是将 $x+y$ 的值转换成整型。

【例 2.5】 强制类型转换实例。

```

1 #include <stdio.h>
2 void main()
3 {
4     float x;
5     int i;
6     x=4.5f;
7     i=(int)x; /*将 x 临时强制转换成整型,离开本行 x 还是单精度型*/
8     printf("x=%f,i=%d\n",x,i);
9 }

```

程序运行结果:

$x=4.500000, i=4$

由此可以看出,有两种类型转换,一种是在运算时不必由用户指定,系统自动进行的类型转换,另一种是强制类型转换。当自动类型转换不能实现目的时,可以用强制类型转换。

例如,当 x 为实型,进行求模运算时 $(int)x\%3$, 必须将 x 强制转换成整型。

2.3 扩展知识与理论

2.3.1 位运算符和位运算

前面所涉及的运算的最小单位为字节,即操作对象的最小单位为字节。

位运算是指进行二进制位的运算,即运算对象不是字节(byte),而是二进制位(bit)。

例如,将一个存储单元中的各二进制位左移或右移一位,两个运算对象按位相“与”等。

C语言提供了6种位运算符,如表2.8所示。

表 2.8 位运算符

运算符	说明	运算符	说明
&	按位“与”	~	按位取反
	按位“或”	<<	左移
^	按位“异或”	>>	右移

注意以下两点:

- ① 位运算符中除“~”以外,均为双目运算符,即要求两侧各有一个运算对象;
- ② 运算对象只能是整型或字符型的数据,不能为实型数据。

1. 按位“与”运算符(&)

参加运算的两个数据,按二进制位进行“与”运算。

运算规则: $0&0=0; 0&1=0; 1&0=0; 1&1=1$ 。

两位同时为“1”,结果才为“1”,否则为0。

例如,求 $3&5$ 的值。

	3=0000011
(&)	5=0000101
	1=0000001

因此, $3&5$ 的值为1。

负数按补码形式参加按位“与”运算。

“与”运算的特殊用途如下。

- (1) 清零。如果想将一个单元清零,即使其全部二进制位为0,只要与一个各位都为

零的数值相与,则其结果为零。

3=00000011
(&) 0=00000000
0=00000000

(2) 取一个数中指定位。

方法:找一个数,对应 x 要取的位,该数的对应位为 1,其余位为零,此数与 x 进行“与”运算可以得到 x 中的指定位。

例如,设 $x=10101110$,取 x 的低 4 位。

x=10101110
(&) 00001111
00001110

取 x 的 2、4、6 位。

x=10101110
(&) 01010100
00000100

2. 按位“或”运算符(|)

参加运算的两个对象,按二进制位进行“或”运算。

运算规则:0|0=0;0|1=1;1|0=1;1|1=1。

参加运算的两个对象只要有一个为 1,其值为 1。

例如,求 3|5 的值。

3=00000011
() 5=00000101
7=00000111

因此,3|5 的值为 7。

负数按补码形式参加按位“或”运算。

“或”运算的特殊用途:常用来对一个数据的某些位置 1。

方法:找到一个数,对应 x 要置 1 的位,该数的对应位为 1,其余位为零,此数与 x 相或可使 x 中的某些位置 1。

例如,将 $x=10100000$ 的低 4 位置 1。

10100000
() 00001111
10101111

3. 按位“异或”运算符(^)

参加运算的两个数据,按二进制位进行“异或”运算。

运算规则:0^0=0;0^1=1;1^0=1;1^1=0。



参加运算的两个对象,如果两个相应位为“异”(值不同),则该位结果为1,否则为0。例如,求 3^5 的值。

	3=0000011
(^)	5=0000101
	6=0000110

因此, 3^5 的值为6。

负数按补码形式参加按位“异或”运算。

“异或”运算的特殊用途如下。

(1) 使特定位翻转。找一个数,对应 x 要翻转的各位,该数的对应位为1,其余位为零,再将此数与 x 对应位“异或”即可。

例如, $x=10101110$,使 x 低4位翻转,高4位不变。

	10101110
(^)	00001111
	10100001

(2) 与0相“异或”,保留原值。从上面的例题可以清楚地看到这一点。

4. 按位取反运算符(~)

参加运算的一个数据,按二进制位进行取反运算。

运算规则: $\sim 1=0$; $\sim 0=1$ 。

对一个二进制数按位取反,即将0变1,1变0。

(~)	0101000010101110
	1010111101010001

使一个数的最低位为零,可以表示为: $a \& \sim 1$ 。

~ 1 的值为1111111111111110,再按“与”运算,最低位一定为0。因为“~”运算符的优先级比算术运算符、关系运算符、逻辑运算符和其他运算符的都高。

5. 左移运算符(<<)

将一个运算对象的各二进制位全部左移若干位(左边的二进制位丢弃,右边补0)。

例如, $a=a \ll 2$ 将 a 的二进制位左移2位,右补0,如图2.14所示。

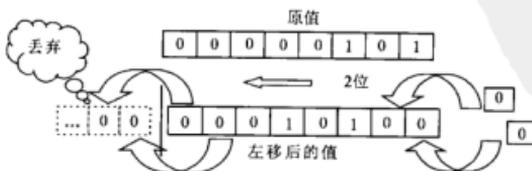


图 2.14 a 左移 2 位示意图

想一想,左移 2 位后 a 的值发生了怎样的变化? a 由 5 变成了 20。若左移时舍弃的高位不包含 1,则每左移一位,相当于该数乘以 2。

6. 右移运算符(>>)

将一个数的各二进制位全部右移若干位,正数左补 0,负数左补 1,右边丢弃。

例如, $a=a>>2$ 将 a 的二进制位右移 2 位,如图 2.15 所示。

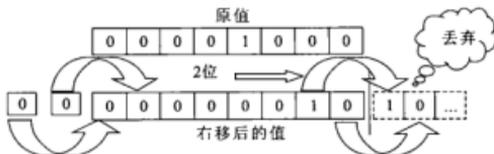


图 2.15 a 右移 2 位示意图

左补 0,还是补 1? 得看被移数是正还是负。

想一想,右移 2 位后 a 的值发生了怎样的变化? a 由 8 变成了 4。操作数每右移一位,相当于该数除以 2。

7. 复合赋值运算符

位运算符与赋值运算符结合,组成新的复合赋值运算符,如下所示。

- (1) $\&=$, 例如, $a\&=b$, 相当于 $a=a\&b$ 。
- (2) $|=$, 例如, $a|=b$, 相当于 $a=a|b$ 。
- (3) $>>=$, 例如, $a>>=b$, 相当于 $a=a>>b$ 。
- (4) $<<=$, 例如, $a<<=b$, 相当于 $a=a<<b$ 。
- (5) $\^=$, 例如, $a\^=b$, 相当于 $a=a\^b$ 。

这些运算符的运算规则与前面讲的复合赋值运算符的运算规则相似。

2.3.2 常见错误及处理方法

常见错误 1: 不明白定义变量的目的。初学 C 语言的学习者,往往对定义变量的目的不很清楚。定义变量就是为了在内存中开辟空间,开辟空间的目的是为了存储数据。

常见错误 2: 混淆数学中的等号与 C 语言赋值运算符的区别。赋值运算符“=”和数学中的等号不同,赋值运算没有相等关系,不能理解成数学中的等式。赋值运算符和它的名称一样,只进行赋值操作,即将表达式(常量、变量)的值赋给左边的变量,因此赋值运算的左边运算对象只能是变量,所以将赋值运算符左边的运算对象称为“左值”。复合赋值运算符与赋值运算符一样,运算符的左边也必须是“左值”。

常见错误 3: 混淆数学运算式与 C 语言表达式的区别。C 语言中的表达式与数学中的运算式不同。如 C 语言中的乘号用星号“*”,C 语言中的乘号不能省略,如数学中的 b^2-4ac ,C 语言中的表达式应写成 $b*b-4*a*c$ 。

常见错误 4: C 语言中的大括号、中括号的含义与数学中的不一样,它们不是改变运算次序的运算符。小括号可以改变运算次序。并且所有括号必须成对使用。

2.4 深入训练

(1) 表达式 $8/4 * (\text{int})2.5 * (\text{int})(1.25 * (3.7 + 2.3))$ 值的数据类型是什么?

(2) 假设所有变量为整型, 编程实现求下列表达式的值。

$x=3, y=6, x++, y++, x+y$

(3) 若 x 和 n 均是整型, 且 x 的初值为 12, n 的初值为 5, 编程实现求下列表达式的值和 x 的值。

$x\%=(n\%2)$

(4) 已知一个英文字符, 编写一个 C 程序, 在屏幕上显示出其前后相连的三个字符。

(5) 设计一段程序, 实现对两个三角形面积的求和。

(6) 设计一程序, 已知三角形三边的长度, 如 $a=3, b=4, c=5$, 按公式

$\text{area} = \sqrt{s(s-a)(s-b)(s-c)}$, 其中 $s=1/2 * (a+b+c)$

计算三角形的面积。

(7) 已知一名学生的五门课考试成绩, 求他的平均成绩。

(8) 设 $x=2.5, k=7, y=4.7, a=2, b=3, c=3.5, d=2.5$ 。编程计算下列表达式的值。

① $x+k\%3 * (\text{int})(x+y)\%2/4$

② $(\text{float})(a+b)/2 + (\text{int})c\%(\text{int})d$

(9) 编程实现英文字符的大小写转换。

(10) 用位运算实现一个数的乘 2 和除 2 操作。

习 题 2

2.1 填空题

(1) 在计算机中, 数据既有_____特性, 又有_____特性。

(2) C 语言的数据基本类型分为_____、_____、_____、_____。

(3) 整型常量的三种表示方法为_____、_____和_____, 实型常量的两种表示方法为_____和_____。

(4) 负数在计算机中以_____形式存储, 将负数转换成补码的步骤是_____, _____, 最后_____。

(5) 声明整型、长整型、单精度型、双精度型、字符型变量后系统在内存中分别开辟的字节空间数依次为_____、_____、_____、_____、_____。

(6) 执行 $\text{int } x=4, y; y=x--$; 后, x 的值是_____, y 的值是_____。

(7) 执行 $\text{int } x=5, y; y=++x$; 后, x 的值是_____, y 的值是_____。

(8) 利用操作符++, 两语句 $k=k+1; f=k;$ 的功能可以由一个语句完成, 这个语句是_____。

(9) 利用操作符 $++$, 两语句 $f=k; k=k+1$; 的功能可以由一个语句完成, 这个语句是_____。

(10) 利用操作符 $--$, 两语句 $k=k-1; f=k$; 的功能可以由一个语句完成, 这个语句是_____。

(11) 利用操作符 $--$, 两语句 $f=k; k=k-1$; 的功能可以由一个语句完成, 这个语句是_____。

(12) 执行 $\text{int } x=4, y=5; y--x--$; 后, x 的值是_____, y 的值是_____。

(13) 执行 $\text{int } x=5, y=4; y++x$; 后, x 的值是_____, y 的值是_____。

(14) 执行 $\text{int } x=4, y; y=x--+3$; 后, x 的值是_____, y 的值是_____。

(15) 执行 $\text{int } x=4, y; y++x-3$; 后, x 的值是_____, y 的值是_____。

(16) 执行 $\text{int } x=6, y; y=x++x++$; 后, x 的值是_____, y 的值是_____。

(17) 执行 $\text{int } x=6, y; y++x+x++$; 后, x 的值是_____, y 的值是_____。

(18) 利用操作符 $++$, 两语句 $k=k+1; f=k+10$; 的功能可以由一个语句完成, 这个语句是_____。

(19) 利用操作符 $--$, 两语句 $f=k-50; k--1$; 的功能可以由一个语句完成, 这个语句是_____。

(20) 在除法(/)运算中两操作数的类型都是整数, 其商为_____, 两操作数至少有一个是实型, 其商为_____。在求模(%)运算中的两个操作数的类型一定要是_____。

(21) 逗号表达式“ $x=5, ++x, x--, x*3$ ”的值是_____。

(22) 字符串“字符串”占用_____字节的空间。

2.2 判断题(判断下列叙述的正确性, 正确的请打“√”, 错误的请打“×”)

- (1) 算术运算符的优先级高于任何一个关系运算符的优先级。 ()
- (2) 任何关系运算符的优先级都比任何逻辑运算符的优先级高。 ()
- (3) C 语言的运算符按运算对象的数目可以分为单目、双目和三目运算符三类。 ()
- (4) C 语言规定, 标识符只能由字母、数字和下划线三种字符组成。 ()
- (5) C 语言中, 有时不同类型的变量可以进行混合运算。 ()
- (6) 在赋值表达式中, 赋值号左边的变量和右边的表达式之值的数据类型可以不相同。 ()

2.3 选择题

(1) 以下程序的输出结果是()。

```
main()
{
    int a=12,b=12;
    printf("%d%d\n", --a, ++b);
}
```

A. 10 11 B. 11 13 C. 11 10 D. 11 12

(2) 若 x, i, j 和 k 都是 `int` 型变量, 则执行下面表达式后 x 的值为()。

`x=(i=4, j=16, k=32)`

A. 4 B. 16 C. 32 D. 2

(3) 以下程序的输出结果是()。

```
main()
{
    int y=6, x=3, z=1;
    printf("%d%d\n", (++x, y++), z+2);
}
```

A. 3 4 B. 6 3 C. 4 3 D. 3 3

(4) 在 C 语言中, 合法的字符常量是()。

A. `'\084'` B. `'\x48'` C. `'ab'` D. `'\0'`

(5) 字符型数据在内存中是以()形式存储的。

A. 原码 B. 补码 C. ASCII 码 D. 反码

(6) (多选) 下列表达式中不正确的有()。

A. `px` B. 3 C. `y÷a` D. `(x)`
 E. `((p+q)*(p-q))` F. `5p` G. `p=(q=8)` H. `(p-i)=39`
 I. `23i+5` J. `k/(p+3)`

(7) 有关运算符的正确描述是()。

A. C 语言的运算符有 40 多种, 运算时的副作用很大
 B. 同级运算符遵循从右到左的运算规则
 C. `35+'a'` 的表达式是合法的
 D. `"&"`、`"|"`、`"!"` 没有优先级

(8) 下面四个选项中, 均是合法转义字符的选项是()。

A. `'\''`、`'\\'`、`'\n'` B. `'\1011'`、`'\'`、`'\017'`
 C. `'\018'`、`'\f'`、`'\xab'` D. `'\0'`、`'\101'`、`'\x1f'`

(9) 以下不正确的语句(设有 `int p, q`)是()。

A. `p*=3;` B. `p/=q;` C. `p+=3;` D. `p&&&=q;`

(10) 设 `int m=1, n=2`, 则 `m++==n` 的结果是()。

A. 0 B. 1 C. 2 D. 3

(11) 设 `a=2, b`; 则执行 `b=a==!a`; 语句后, b 的结果是()。

A. 0 B. 1 C. 2 D. 3

(12) `sizeof(double)` 是一个()表达式。

A. 整型 B. 双精度 C. 不合法 D. 函数调用

(13) 以下程序的输出结果是()。

```
main()
{
    int n=1;
```

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

```
}
```

A. 1,2,1

B. 1,2,2

C. 2,3,2

D. 2,2,1

(14) 设 $\text{int } a=2, b=2$, 则 $++a+b$ 的结果是(), a 的结果是(), b 的结果是()。

A. 2

B. 3

C. 4

D. 5



单元3 项目封面与菜单的初步设计

能力目标

- 会用格式化输入(`scanf`)、输出(`printf`)函数输入/输出各种类型的数据。
- 掌握“%*±*md”、“%*±*m.nf”、“%*±*m.n*s*”、“%*c*”等格式指示符的使用方法。
- 会用单个字符输入/输出函数处理单个字符或多个字符的输入/输出。
- 会用输入/输出函数建立“班级学生成绩管理系统”项目中的封面和主、子菜单。

知识目标

- 理解格式化输入/输出函数的格式,理解格式化输入/输出函数的格式指示符的含义。
- 理解负数在内存中的存储特点。
- 理解转义字符在 `printf` 输出函数中的应用,理解转义字符在 `scanf` 输入函数中的特点。
- 理解数值型数据和字符型数据混合输入的要点。

学习提示

语句是 C 语言程序中最基本的单位,程序的运行过程就是执行程序语句的过程。C 语言没有输入/输出语句,只有输入/输出函数,用好、用活 C 语言的输入/输出库函数是学习编程的重要内容。

3.1 任务3:用输入/输出函数初步设计项目封面与菜单

一般软件应该包含软件封面和主、子菜单,“班级学生成绩管理系统”也不例外,也应当设计该软件的封面和主、子菜单。

本任务采用 C 语言的标准输入/输出函数,分别实现“班级学生成绩管理系统”的封面和主、子菜单,这些内容暂时处理成单个程序的形式,后面将介绍如何将这封面与主、子菜单组合起来。

项目封面程序:

```
#include <stdio.h>
#include <stdlib.h>
void main( )           //项目封面
{
    system("cls");//执行 DOS 清屏命令函数
    printf("\n\n\n");
    printf("\t\t 班级学生成绩管理系统\n\n");
    printf("\t\t 版本号:1.0\n\n");
    printf("\n\n\n\n");
    printf("\t\t 2008 年 5 月\n\n");
    printf("\t\t 程序设计兴趣小组\n\n");
}
```

项目主菜单程序:

```
#include <stdio.h>
#include <stdlib.h>
void main( )           //项目主菜单
{
    int n;
    system("cls");
    printf("\n\n\n");
    printf("          |* * * * * 学生成绩管理系统* * * * * |\n");
    printf("          |.....|\n");
    printf("          |           请选择菜单序号(0~6)           |\n");
    printf("          |.....|\n");
    printf("          |           1----打开文件           |\n");
    printf("          |           2----保存文件           |\n");
    printf("          |           3----编辑数据           |\n");
    printf("          |           4----显示数据           |\n");
    printf("          |           5----数据计算           |\n");
    printf("          |           6----程序说明           |\n");
    printf("          |           0----退出系统           |\n");
    printf("          |.....|\n");
    printf("\t\t 请选择序号:");
    scanf("%d",&n);
    printf("您选择了第%d项!\n",n);
}
```

项目编辑子菜单程序:

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

```

void main()//项目编辑子菜单
{
    int n;
    system("cls");
    printf("\n\n\n");
    printf("          |* * * * * 编辑子菜单* * * * * |\n");
    printf("          |..... |\n");
    printf("          |          请选择菜单序号(0~3)          |\n");
    printf("          |..... |\n");
    printf("          |          1----增加记录          |\n");
    printf("          |          2----删除记录          |\n");
    printf("          |          3----修改记录          |\n");
    printf("          |          0----返回上级菜单        |\n");
    printf("          |..... |\n");
    printf("\t\t\t 请选择序号:");
    scanf("%d",&n);
    printf("您选择了第%d项!\n",n);
}

```

项目查看子菜单程序:

```

#include <stdio.h>
#include <stdlib.h>
void main()//项目查看子菜单
{
    int n;
    system("cls");
    printf("\n\n\n");
    printf("          |* * * * * 查看子菜单* * * * * |\n");
    printf("          |..... |\n");
    printf("          |          请选择菜单序号(0~4)          |\n");
    printf("          |..... |\n");
    printf("          |          1----查看选定记录          |\n");
    printf("          |          2----显示全部记录          |\n");
    printf("          |          3----显示排序记录          |\n");
    printf("          |          4----显示不及格记录        |\n");
    printf("          |          0----返回上级菜单        |\n");
    printf("          |..... |\n");
    printf("\t\t\t 请选择序号:");
    scanf("%d",&n);
    printf("您选择了第%d项!\n",n);
}

```

```

}
项目计算子菜单程序:
#include <stdio.h>
#include <stdlib.h>
void main() //项目计算子菜单
{

```

```

    int n;
    system("cls");
    printf("\n\n\n");
    printf("        | * * * * * 计算子菜单 * * * * * |\n");
    printf("        | ..... |\n");
    printf("        |           请选择菜单序号(0~3)           |\n");
    printf("        | ..... |\n");
    printf("        |     1----计算总成绩和平均成绩     |\n");
    printf("        |     2----计算最高分                 |\n");
    printf("        |     3----计算最低分                 |\n");
    printf("        |     0----返回上级菜单               |\n");
    printf("        | ..... |\n");
    printf("\t\t 请选择序号:");
    scanf("%d",&n);
    printf("您选择了第%d项!\n",n);
}

```

```

项目排序子菜单程序:
#include <stdio.h>
#include <stdlib.h>
void main() //项目排序子菜单
{

```

```

    int n;
    system("cls");
    printf("\n\n\n");
    printf("        | * * * * * 排序子菜单 * * * * * |\n");
    printf("        | ..... |\n");
    printf("        |           请选择菜单序号(0~2)           |\n");
    printf("        | ..... |\n");
    printf("        |     1----按升序排序                 |\n");
    printf("        |     2----按降序排序                 |\n");
    printf("        |     0----返回上级菜单               |\n");
    printf("        | ..... |\n");
    printf("\t\t 请选择序号:");

```

```
scanf("%d",&n);
printf("您选择了第%d项!\n",n);
}
```

在上述程序中使用了 system 函数,这个函数是一个库函数,它能发出一个 MS-DOS 命令,括号中的“cls”是 MS-DOS 的清屏命令,通过 system 函数执行了一条 DOS 命令。

单元能力训练任务分别如下。

任务 A:正确调用格式化输入/输出函数输入/输出不同类型的数据。

任务 B:正确进行不同类型数据的混合输入。

任务 C:设计“班级学生成绩管理系统”封面和主、子菜单。

任务 D:模仿任务 3 设计“学生通讯录”封面和主、子菜单。

3.2 必备知识与理论

3.2.1 C 语句

C 语言的语句向计算机系统发出操作指令,用来完成一定的操作任务,一条语句经编译后产生若干条机器指令,只有语句才能转变成指令。一个实际的程序应当包含若干语句,但一个程序不都是语句,前面讲过一个函数包含声明部分和执行部分,声明部分不是 C 语句,执行部分才是 C 语句,当然预编译命令也不是 C 语句。

C 语句可以分为以下 5 类。

1. 控制语句

控制语句能完成一定的控制功能,如分支程序的控制、循环控制等。

2. 表达式语句

表达式语句指由一个表达式构成的一条语句,如由赋值表达式构成的语句。如:

```
a=3; /*是一个赋值语句*/
```

赋值表达式与赋值语句有什么区别呢?其实只要在赋值表达式后面加上分号,赋值表达式就变成赋值语句了。 $a=3$ 为表达式, $a=3;$ 为语句。

任何表达式都可以加上分号而成为语句,分号是语句不可缺少的一部分,也是区别表达式与语句的重要标志。如:

```
i++; /*是一条语句,作用是使 i 值加 1*/
```

```
a+b; /*是一条语句,作用是完成 a+b 的操作,但其和没有送给任何变量*/
```

由于赋值语句应用十分广泛,以下进一步说明。

C 语句中的赋值符“=”是一个运算符,与数学中的“等号”是有区别的,赋值没有相等关系。

3. 函数调用语句

函数调用语句由一次函数调用后跟一个分号构成一条语句。如：

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

注意：函数调用后面有一个分号，它是C语言中语句的标志。

4. 空语句

只有一个分号也是语句，它是空语句。这个语句表示什么都不做。如：

```
； /*空语句*/
```

这样的语句有实际意义吗？有！大多数情况下用作转向点或作为循环语句中的循环体。

5. 复合语句

用大括号{ }将一些语句括起来的语句称为复合语句。如：

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

3.2.2 格式化输入/输出函数

1. 格式化输出函数(printf 函数)

printf 函数的作用：向计算机系统默认的输出设备（一般指显示器）输出一个或多个任意类型的数据。

【例 3.1】 已知圆半径 radius=1.5，求圆周长和圆面积。

```
1 #include <stdio.h>  
2 void main()  
3 {  
4     float radius,length,area,pi=3.1415926;  
5     radius=1.5;  
6     length=2*pi*radius; /*求圆周长*/  
7     area=pi*radius*radius; /*求圆面积*/  
8     printf("radius=%f\n",radius); /*输出圆半径*/  
9     printf("length=%7.2f,area=%7.2f\n",length,area);  
    //输出圆周长和面积  
10 }
```

程序运行结果：

```
radius=1.500000
length= 9.42,area= 7.07
```

1) printf 函数格式

格式:

```
printf("格式字符串"[,输出项表]);
```

(1) 格式字符串。“格式字符串”也称“转换控制字符串”，是用双引号括起来的字符串，包括以下三种字符串。

① 格式指示符。由“%”和类型转换字符组成，如%d、%f等，它的作用是将输出的数据转换为指定的格式输出。格式说明总是由“%”字符开始。

② 转义字符。例如，例3.1中printf()函数中的‘\n’就是转义字符，输出时产生一个“回车换行”操作。

③ 普通字符。除格式指示符和转义字符之外的其他字符。格式字符串中的普通字符，输出时按原样输出。

例如，例3.1中第8行，输出函数各部分的含义如图3.1所示。

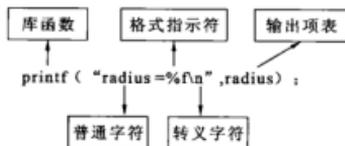


图3.1 输出函数各部分的含义示意图

(2) 输出项表。输出项表是可选的。如果要输出的数据不止1个，相邻2个之间用逗号分开。下面的printf()函数都是合法的：

```
printf("I am a student.\n");
printf("%d",3+2);
printf("a=%f b=%5d\n", a, a+3);
```

必须强调的是，“格式字符串”中的格式指示符必须与“输出项表”中输出项的数据类型一致，且个数相同，否则会引起输出错误。

2) 类型转换字符

输出不同类型的数据，要使用不同的类型转换字符。

(1) 类型转换字符d——以带符号的十进制整数形式输出格式符。

【例3.2】 类型转换字符d的使用。

```
1 #include <stdio.h>
2 void main()
3 {
4     int num1=123;
```

```

5 long num2=123456;
6 printf("num1=%d,num1=%5d,num1=%-5d,num1=%2d\n",
    num1,num1,num1,num1); //用4种不同格式,输出int型数据num1
    的值
7 printf("num2=%ld,num2=%8ld,num2=%5ld\n",
    num2,num2,num2); //用3种不同格式,输出long型数据num2的值
8 )

```

程序运行结果:

```

num1=123,num1=□□123,num1=123□□, num1=123
num2=123456,num2=□□123456,num2=123456

```

d 格式符用来输出十进制整数,有以下几种形式。

① %d,按整型数据的实际长度输出。

② %±md,按指定数据宽度输出,m是指定的数据宽度,如果数据的位数小于m,则左端补空格(+号)或右端补空格(-号),若大于m,则按实际位数输出。正号表示输出右对齐(也是默认对齐方式),负号表示输出左对齐。

③ %ld,输出长整型数据,如上例中的 num2 都是采用长整型格式输出的。

对于整数,还可用八进制、十六进制、无符号形式输出。

(2) 类型转换字符 o——以八进制整数形式输出格式符。

内存中八进制输出的数值是不带符号的,即将符号位也一起作为八进制数的一部分输出。

请看下面的例子:

```

int a=-1;
printf("%d,%o",a,a);

```

输出结果:

```
-1,3777777777
```

1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1

图 3.2 -1 在内存中的存储示意图

为什么是这样? -1 在内存中的存储情况如图 3.2 所示。

所以按十进制输出为 -1,八进制输出为 3777777777。

(3) 类型转换字符 x——以十六进制整数形式输出格式符。

由于符号位也作为了十六进制数的一部分,同样

不会出现负的十六进制数。例如:

```

int a=-1;
printf("%x,%o,%d",a,a,a);

```

输出结果为:

```
fffffff,3777777777,-1
```

(4) 类型转换字符 u——以无符号形式输出格式符。

用来以十进制整数形式输出 unsigned 型数据。例如:

```
int a=-1;
printf("%u",a);
```

输出结果为:

```
4294967295
```

想一想,为什么是这样一个数据?

(5) 类型转换字符 c——以字符形式输出格式符。

【例 3.3】 类型转换字符 c 的使用。

```
1 #include <stdio.h>
2 void main( )
3 {
4     char c='A';
5     int i=65;
6     printf("c=%c,%5c,%d\n",c,c,c);
7     printf("i=%d,%c",i,i);
8 }
```

程序运行结果:

```
c=A,□□□□A,65
i=65,A
```

注意:%后面的 c 是格式符,逗号后面的 c 是变量,不要搞混淆了,同时还要注意一个字符变量和一个整型变量可以用 %c 或 %d 两种格式输出。

需要强调的是,在 C 语言中,整数可以用字符形式输出,字符数据也可以用整数形式输出。将整数用字符形式输出时,系统首先求该数与 256 的余数,然后将余数作为 ASCII 码,转换成相应的字符输出。

(6) 类型转换字符 s——输出一个字符串。

【例 3.4】 类型转换字符 s 的使用。

```
1 #include <stdio.h>
2 void main( )
3 {
4     printf("%s\n%5s\n%-10s\n","Internet","Internet","Internet");
5     printf("%%10.5s\n%-10.5s\n%3.5s\n","Internet","Internet",
6     "Internet");
```

程序运行结果:

```
Internet
Internet
Internet□□
□□□□□Inter
Inter□□□□□
Inter
```



注意：系统输出字符和字符串时，不输出单引号或双引号。

s 格式符的几种用法如下。

① %s, 原样输出字符串。

② %±ms, 其中“+”号表示右对齐, “-”号表示左对齐, 输出的字符串占 m 列, 如果字符串本身的长度大于 m 值, 则突破 m 的限制, 将字符串原样输出; 如果字符串本身的长度小于 m 值, 不足部分左边补空格(右对齐)或右边补空格(左对齐)。

③ %±m.ns, 其中“+”号表示右对齐, “-”号表示左对齐, 输出的字符串占 m 列, 但字符串只取从左到右 n 个字符, 不足部分左边补空格(右对齐)或右边补空格(左对齐)。如果 n>m, 则 m 自动取 n 值。

(7) 类型转换字符 f——以小数形式输出单精度和双精度实数。

【例 3.5】 类型转换字符 f 的使用。

```
1 #include <stdio.h>
2 void main()
3 {
4     float f=123.456;
5     double d1,d2;
6     d1=1111111111111111.111111111;
7     d2=222222222222222.222222222;
8     printf("%f\n%12f\n%12.2f\n%-12.2f\n%.2f\n", f, f, f, f, f);
9     printf("d1+d2=%f\n", d1+d2);
10 }
```

程序运行结果：

```
123.456001
□□123.456001
□□□□□123.46
123.46□□□□□
123.46
d1+d2=333333333333.333010
```

f 格式符的两种用法如下。

① %f, 不指定字符宽度, 由系统自动指定, 使整数部分全部如数输出, 并输出 6 位小数, 但不要认为全部数字都是有效数字。

本例程序的输出结果中, 数据 123.456001 和 333333333333.333010 中的 001 和 010 都是无意义的, 因为它们超出了有效数字的范围(单精度的有效范围为 7 位数字, 双精度的有效范围一般为 16 位数字)。

② %±m.nf, 其中“+”号表示右对齐, “-”号表示左对齐, 指定输出的数据占 m 列(包括小数点), 其中有 n 位小数, 如果位数不足左边补空格(右对齐)或右边补空格(左对齐)。

2. 格式化输入函数(scanf 函数)

scanf 函数是用来从外部输入设备(键盘)向计算机主机(内存变量)输入数据。

【例 3.6】 从键盘上输入圆柱体的底面半径 radius、高 high 的值，求其体积。

```

1 #include <stdio.h>
2 #define PI 3.1415926 //定义字符常量
3 void main()
4 {
5     float radius,high,vol;
6     printf("Please input radius and high: ");
7     scanf("%f%f",&radius,&high);/*从键盘输入两个实数赋给变量
   radius,high*/
8     vol=PI*radius*radius*high;
9     printf("radius=%7.2f\nhigh=%7.2f\nvol=%7.2f\n",radius,
   high,vol);
10 }

```

程序运行结果：

```

Please input radius and high: 1.5□2.0<回车>
radius=□□□1.50
high=□□□2.00
vol=□□13.14

```

在 C 语言中，可用两种方法给程序提供数据，一种是用赋值语句，另一种就是用 scanf 函数，通过键盘输入，给程序提供数据，使程序变得更为灵活。

1) scanf 函数格式

格式：

```
scanf("格式字符串",输入项首地址表);
```

(1) 格式字符串。格式字符串可以包含 3 种类型的字符：格式指示符、空白字符（空格、Tab 键和回车键）和非空白字符（又称普通字符）。

格式指示符与 printf() 函数相似，空白字符作为相邻 2 个输入数据的缺省分隔符，非空白字符在输入有效数据时，必须原样输入。

(2) 输入项首地址表。由若干个输入项首地址组成，相邻 2 个输入项首地址之间用逗号分开。

输入项首地址表中的地址可以是变量的首地址，也可以是数组名。

变量首地址的表示方法：**&** 变量名。

其中“&”是地址运算符，也可以称为取地址符。例如，例 3.6 中的“&radius”是指变量 radius 在内存中的首地址。

2) 格式指示符

输入格式指示符和 printf 函数的格式指示符相似。

例如：

```
scanf("%3c%3c",&ch1,&ch2);  
printf("ch1=%c,ch2=%c\n",ch1,ch2);
```

假设输入“abcdefg”，则系统将读取的“abc”中的“a”赋给变量 ch1；再读取“def”中的“d”赋给变量 ch2，所以 printf 函数的输出结果为：ch1=a,ch2=d。

3) 数据输入操作

(1) 如果相邻两个格式指示符之间，不指定分隔符(如逗号、冒号等)，则相应的两个输入数据之间，至少用一个空格分开，或者用 Tab 键分开，或者输入一个数据后，按回车键，然后再输入下一个数据。

例如，scanf("%d%d",&num1,&num2)；

假设将 12 输入给 num1,36 输入给 num2,则正确的输入操作为：12□36<回车>

或者：12<回车>

36<回车>

(2) “格式字符串”中出现的普通字符(包括转义字符)，务必原样输入。

例如，scanf("%d,%d",&num1,&num2)；

在两个 %d 之间有一个逗号，是普通字符，正确的输入操作为：

12,36<回车>

另外，scanf 函数中，格式字符串内的转义字符(如 \n)，系统并不把它视为转义字符来解释，不会产生一个控制操作，而是将其视为普通字符，所以也要原样输入。

例如，scanf("num1=%d,num2=%d\n",&num1,&num2)；

正确的输入操作为：num1=12,num2=36\n<回车>

提高人机交互性建议：为改善人机交互，同时简化输入操作，在设计输入操作时，一般先用 printf 函数输出一个提示信息，再用 scanf 函数进行数据输入。

例如，将 scanf("num1=%d,num2=%d\n",&num1,&num2)；

改为：

```
printf("num1=");scanf("%d",&num1);
```

```
printf("num2=");scanf("%d",&num2);
```

这样可以改善用户界面，做到界面友好，提高程序的可操作性。

(3) 输入数据时，遇到以下情况，系统认为该数据结束。

遇到空格、回车键，或者 Tab 键。

遇到输入域宽度结束。如“%3d”，只取 3 列。

遇到非法输入。例如，在输入数值数据时，遇到字母等非数值符号(数值符号仅由数字字符 0~9、小数点和正负号构成)。

(4) 使用格式说明符“%c”输入单个字符时，空格和转义字符均作为有效字符被接受。

例如：

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

```
printf("ch1=%c,ch2=%c,ch3=%c\n",ch1,ch2,ch3);
```

假设输入：A□B□C，则系统将字母‘A’赋值给 ch1，空格‘□’赋值给 ch2，字母‘B’赋值给 ch3。正确的输入方法应当是 ABC。

(5) 数值型数据与字符型数据混合输入。

数值型数据(整型、实型)与字符型数据的混合输入时要特别小心,如果输入格式不对就不能得到正确的结果。

例如:

```
int x1,x2;
char c1,c2;
scanf("%d%c%d%c",&x1,&c1,&x2,&c2);
printf("%d,%c,%d,%c\n",x1,c1,x2,c2);
```

如果输出结果为:10,a,20,b

正确的输入方法为:10a20b<回车>

即数字与字符之间不能有空格。这是因为空格是字符,如果 10 与 a 之间输入了空格,系统将空格送给 c1,而不是将字符 a 送给 c1。

printf 函数和 scanf 函数都是 C 语言的库函数,使用它们时,不要忘记在文件开头加上编译预处理命令: #include <stdio.h>。

3.3 扩展知识与理论

3.3.1 单个字符输入/输出函数

如果把格式化输入/输出函数比喻为输入/输出的全能选手,单个字符输入/输出函数就可以比喻为单项选手。因为,它只能输入/输出字符数据。

1. 单个字符输出函数(putchar 函数)

【例 3.7】 putchar 函数的格式和使用方法。

```
1 #include "stdio.h"
2 void main( )
3 {
4     char ch1='N', ch2='E', ch3='W';
5     putchar(ch1); putchar(ch2); putchar(ch3); /*输出*/
6     putchar('\n'); /*换行*/
7     putchar(ch1); putchar('\n'); /*输出 ch1 的值,并换行*/
8     putchar('E'); putchar('\n'); /*输出字符'E',并换行*/
9     putchar(ch3); putchar('\n'); /*输出 ch3 的值,并换行*/
10 }
```

程序运行结果:

```
NEW
N
E
```

w

1) putchar 函数格式

```
putchar(ch);
```

其中 ch 可以是一个字符变量或常量,也可以是一个转义字符。

2) putchar 函数功能

向终端(屏幕)输出一个字符。

putchar 函数只能用于单个字符的输出,且一次只能输出一个字符。另外,从功能角度来看,printf 函数完全可以代替 putchar 函数。

2. 单个字符输入函数(getchar 函数)

【例 3.8】 说明 getchar 函数的格式和作用。

```
1 #include "stdio.h"    /*文件包含*/
2 void main()
3 {
4     char ch;
5     printf("Please input two character: ");
6     ch=getchar();      /*输入 1 个字符并赋给 ch*/
7     putchar(ch);putchar('\n');
8     putchar(getchar()); /*输入 1 个字符并输出*/
9     putchar('\n');
10 }
```

程序运行结果:

```
Please input two characters: ab<回车> (注意输入方法)
a
b
```

1) getchar 函数格式

```
getchar();
```

2) getchar 函数功能

从键盘向变量输入一个字符。另外,从功能角度来看,scanf 函数完全可以代替 getchar 函数。

getchar 函数只能用于单个字符的输入,一次输入一个字符。

例 3.8 中程序的功能是输入一个字符,显示一个字符,回车换行,再输入并显示一个字符。而运行时字符是连续输入的,运行结果却是正确的,这是因为输入字符后,它们暂存于键盘的缓冲区中,然后由 getchar 函数从键盘缓冲区中一个一个地取出来。

putchar 函数和 getchar 函数也是 C 语言库函数,使用时必须在文件的开头加上编译预处理命令: #include <stdio. h>。

3. 输入一个字符无回显函数(getch 函数)

1) getch 函数格式

```
getch();
```

2) getch 函数功能

从键盘上输入一个字符,它在屏幕上并不显示该字符。它往往用来使屏幕暂停和输入密码等操作,具体实例请看后续内容。

getch 函数也是 C 语言的库函数,使用时必须在文件的开头加上 #include <conin. h>。

3.3.2 常见错误及处理方法

常见错误 1:使用 C 语言的库函数时忘记在文件的开头加上使用库函数必须的编译预处理命令。

常见错误 2: int x; scanf("%d", x);

初学 C 语言程序设计的读者,在使用格式化输入函数时,一个最常出现的错误是,忘记在输入项首地址表的变量前加上取地址符“&”,即在 scanf 函数中用的是变量而不是变量地址,这样编译时系统就会提示变量没有赋初值的错(“local variable ‘x’ used without having been initialized”)。

常见错误 3:格式化输入函数使用了不必要的普通字符。如:scanf("%d\n",&n);。

输入函数中的‘\n’就是不必要的普通字符,在输入函数中转义字符也当普通字符处理,输入时要原样输入,这样就会使学习者难以做到正确输入。

常见错误 4:使用格式化输入/输出函数时,格式指示符与输入/输出项在个数、类型和顺序上不一致。修改的方法是将格式指示符与输入/输出项在个数、类型和顺序上调整为一致。

3.4 深入训练

- (1) 输出 4 行 4 列星号,使之排列成矩形。
- (2) 在屏幕上输出下面的图形。

```

      *
     * *
    *  *
   *  *
  *

```




```
scanf("%d%c%d%c", &x1, &ch1, &x2, &ch2);
```

```
printf("%d,%c,%d,%c", x1, ch1, x2, ch2);
```

- A. 40 x 50 y B. 40,x,50,y C. 40x50y D. 4050xy

(3) 以下程序段的执行结果是()。

```
float f=13.8f;
```

```
int n;
```

```
n= ((int)f)%3;
```

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

- A. n=2 B. n=1 C. 2 D. 1

(4) 以下正确的叙述是()。

- A. 在C程序中,每行中只能写一条语句。
 B. 若a是实型变量,C程序中允许赋值a=10,因此实型变量中允许存放整型数据。
 C. 在C程序中,无论是整数还是实数,都能被准确无误地表示。
 D. 在C程序中,%是只能用于整数运算的运算符。

(5) 以下说法正确的是()。

- A. 输入项可以为一个实型常量,如scanf("%f",3.5);
 B. 只有格式控制,没有输入项,也能进行正确输入,如scanf("a=%d,b=%d");
 C. 当输入一个实型数据时,格式控制部分应规定小数点的位数,如scanf("%4.2f",&f);
 D. 当输入数据时,必须指明变量的地址,如scanf("%f",&f);

(6) 根据下面的程序及数据的输入和输出形式,程序中输入语句的正确形式应该为()。

```
定义变量:char ch1,ch2,ch3;
```

```
输出语句:printf("%c%c%c",ch1,ch2,ch3);
```

```
输入形式:ABC
```

```
输出形式:ABC
```

- A. scanf("%c%c%c",&ch1,&ch2,&ch3);
 B. scanf(" %c,%c,%c",&ch1,&ch2,&ch3);
 C. scanf(" %c %c %c",&ch1,&ch2,&ch3);
 D. scanf(" %c%c",&ch1,&ch2,&ch3);

(7) 有输入语句:scanf(" a=%d,b=%d,c=%d",&a,&b,&c);为使变量a的值为1,b为3,c为2,从键盘输入数据的正确形式应当是()。

- A. 132<回车> B. 1,3,2<回车>
 C. a=1 b=3 c=2<回车> D. a=1,b=3,c=2<回车>

(8) 若运行时给变量x输入12,则以下程序的运行结果是()。

```
main( )
```

```
{
```

```
int x,y;
```

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

```
y=x++;
```

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

```
}
```

A. x=13,y=12

B. x=13,y=13

C. 13,12

D. 13,13

(9) 以下程序的运行结果是()。

```
main()
```

```
{
```

```
int a=4,b=5;
```

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

```
}
```

A. 10,5

B. 9,5

C. 9,4

D. 结果错

(10) 设 $a=3, b=4, c=5$, 则逻辑表达式 $!(a+b)+c-1 \&\& b+c/2$ 的值为()。

A. 0

B. 1

C. 2

D. -1



单元4 项目封面、菜单的顺序执行设计

能力目标

- 掌握顺序结构程序的编写方法。
- 会画顺序结构流程图或 N-S 图。
- 能编写将“班级管理信息系统”封面和主、子菜单连接起来的程序。

知识目标

- 进一步理解 C 语言程序中的语句。
- 理解顺序结构程序的执行顺序与方法。

学习提示

程序中最基本、最简单、最常用的结构就是顺序结构,任何一种程序都离不开顺序结构。学习顺序结构时,一定要注意顺序结构的执行特点。

4.1 任务 4:项目封面、菜单的顺序执行设计

该任务将封面和主、子菜单连接起来,实现封面、菜单的顺序执行。由于一个程序只有一个主函数,因此,将封面、主菜单、编辑子菜单、查看子菜单、计算子菜单、排序子菜单程序中的主函数分别改名为 StuCover、MainMenu、EditMenu、DispMenu、CompMenu、SortMenu,并将这些函数复制到一个 C 语言程序中,另外,再建立一个主函数,分别执行这些函数。

由于上述 6 个函数内容与任务 3 相同,为了节约书写的空间,除主函数之外,其他函数只给出函数名。

```
void StuCover() //项目封面函数
{
}
void MainMenu() //项目主菜单函数
{
}
```

```
void EditMenu() //项目编辑子菜单函数
{
}
void DispMenu() //项目查看子菜单函数
{
}
void CompMenu() //项目计算子菜单函数
{
}
void SortMenu() //项目排序子菜单函数
{
}
void main() //项目主函数
{
    StuCover(); //函数调用语句,执行封面函数
    getch(); //屏幕暂停函数
    MainMenu(); //函数调用语句,执行主菜单函数
    getch();
    EditMenu(); //函数调用语句,执行编辑子菜单函数
    getch();
    DispMenu(); //函数调用语句,执行查看子菜单函数
    getch();
    CompMenu(); //函数调用语句,执行计算子菜单函数
    getch();
    SortMenu(); //函数调用语句,执行排序子菜单函数
    getch();
}
```

要注意的是,主函数是放在所有其他函数后面的。能否将主函数放到所有函数前面或者主函数的位置是否可以任意呢?回答是肯定的,关于如何实现这一点我们将在后续内容中讲解。

该任务使用了一个 `getch` 库函数,它在这里的作用是使屏幕发生暂停,当程序执行到此函数时发生暂停,等待用户输入一个任意字符后,程序继续向下执行。这样做的好处是,可以使用户看清封面和主、子菜单。

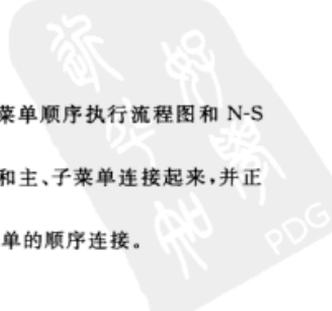
单元能力训练任务分别如下。

任务 A: 正确完成顺序结构程序的设计。

任务 B: 画出“班级学生成绩管理系统”中封面和主、子菜单顺序执行流程图和 N-S 图。

任务 C: 用顺序结构将“班级学生成绩管理系统”的封面和主、子菜单连接起来,并正确执行。

任务 D: 模仿任务 4 完成“学生通讯录”的封面和主、子菜单的顺序连接。



4.2 必备知识与理论

4.2.1 顺序结构程序设计

C语言规定,无论主函数在什么位置,程序执行总是从主函数开始。任务4有一个特点,就是程序从主函数进入后,是按照语句的书写顺序,依次执行的,这种程序结构就是顺序结构。

【例 4.1】 输入任意三个整数,求它们的和与平均值。

```
1 #include "stdio.h"
2 void main()
3 {
4     int num1,num2,num3,sum;
5     float aver;
6     printf("Please input three numbers:");
7     scanf("%d,%d,%d",&num1,&num2,&num3);/*注意输入格式*/
8     sum=num1+num2+num3; /*求累计和*/
9     aver=sum/3.0f;      /*求平均值*/
10    printf("num1=%d,num2=%d,num3=%d\n",num1,num2,num3);
11    printf("sum=%d,aver=%7.2f\n",sum,aver);
12 }
```

程序运行结果:

```
Please input three numbers: 85,75,95<回车>
num1=85,num2=75,num3=95
sum=255,aver=□□85.00
```

顺序结构程序流程如图 4.1 所示。

想一想:能否将“aver=sum/3.0;”中“3.0”改为“3”,结果会发生变化吗?

【例 4.2】 求方程 $ax^2+bx+c=0$ 的实数根。a、b、c 由键盘输入, $a \neq 0$ 且 $b^2-4ac > 0$ 。

```
1 #include <stdio.h>
2 #include "math.h" //使用求平方根函数 sqrt(),必须包含 math.h 头文件
3 void main()
4 {
5     float a,b,c,disc,x1,x2;
6     printf("Input a, b, c: ");
7     scanf("%f,%f,%f",&a,&b,&c); /*输入方程的三个系数的值*/
8     disc=b*b-4*a*c;           /*求判别式的值赋给 disc*/
```

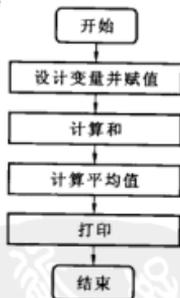


图 4.1 顺序结构程序流程图

```

9      x1=(-b+sqrt(disc))/(2*a);
10     x2=(-b-sqrt(disc))/(2*a);
11     printf("\nx1=%6.2f\nx2=%6.2f\n",x1,x2);
12 }

```

程序运行结果：

```

Input a, b, c: 1,-5,-6 <回车>
x1=□□6.00
x2=□□-1.00

```

【例 4.3】 键盘输入一个小写字母,以大写形式输出该字母及对应的 ASCII 码。

```

1 #include "stdio.h"
2 void main()
3 {
4     char c1,c2;
5     printf("Input a lowercase letter: ");
6     c1=getchar();
7     putchar(c1);printf(",%d\n",c1);
8     c2=c1-32; /*将小写字母转换成对应的大写字母*/
9     printf("%c,%d\n",c2,c2);
10 }

```

程序运行结果：

```

Input a lowercase letter: a<回车>
a,97
A,65

```

4.2.2 顺序结构特点

顺序结构的最大特点是,程序的每条语句都会被执行到且只执行一次。这种程序结构的执行顺序与语句的书写顺序有关,写在前面的先执行,写在后面的后执行,没有不被执行的语句。

一般的顺序结构程序包括以下几个部分:

- ① 变量类型的说明;
- ② 提供数据语句;
- ③ 运算部分语句;
- ④ 输出部分语句。

当然,不同目的的程序,有些部分也可能不一样。

4.3 深入训练

- (1) 编程实现从键盘输入三角形的三条边,计算三角形面积。



(2) 编写程序实现输入一个字符组成一个“中”字,如输入字符 B,图案为:

```
      B
    BBBBBBBBBB
  B   B   B
  B   B   B
    BBBBBBBBBB
      B
      B
```

(3) 编程实现输入五个学生的成绩,计算其总成绩和平均成绩。

(4) 编写程序解决鸡与兔同笼问题:已知鸡兔总数为 a,鸡兔腿总数为 b,计算出鸡兔各多少只。

习 题 4

4.1 填空题

(1) 借助于临时变量 k 交换 a 和 b 两个变量的值,应顺序执行赋值语句是_____、_____和_____。

(2) 以下程序段执行结果是_____。

```
main ( )
{
    short x=-1;
    printf("%d,%o,%x,%u\n",x,x,x,x);
}
```

(3) 以下程序段执行结果是_____。

```
main ( )
{
    float f=3.1415926f;
    printf("%f,%5.2f,%3.3f\n",f,f,f);
}
```

(4) 以下程序的运行结果是_____。

```
#include <stdio.h>
main ( )
{
    int a=-5,b=-3;
    printf("%d",-a%b);
    printf("%d", (a-b,a+b));
}
```



A. 123,45.000000

B. 123,345

C. 45.000000,678.000000

D. 45678.000000,123.000

(5) 若以下变量均是整型,且 $\text{num}=\text{Sum}=7$;则计算表达式 $\text{Sum}=\text{num}++$, $\text{Sum}++$, $++\text{num}$; 后 Sum 的值为()。

A. 7

B. 8

C. 9

D. 10



单元5 项目菜单的选择执行设计

能力目标

- 掌握用 if-else 语句实现分支结构和多分支结构的方法。
- 掌握条件运算符构成分支结构的方法。
- 掌握用 switch 语句实现多分支结构的方法,掌握 break 语句的使用方法。
- 会画分支结构流程图或 N-S 图。
- 能用 if-else 语句和 switch 语句实现“班级学生成绩管理系统”中的主、子菜单的选择执行。

知识目标

- 理解 if 语句和 if-else 语句的定义格式和执行顺序。
- 理解 if-else 语句嵌套的概念与利用嵌套定义实现多分支结构的方法。
- 理解条件运算符构成的语句与 if 语句的相同点与不同点。
- 理解 switch 语句实现多分支结构的原理、定义格式和执行顺序,理解 break 语句的执行过程。

学习提示

在多数情况下只有顺序结构的程序是很少的,一般还包括分支和循环结构,这里主要介绍分支结构。分支结构有两种实现方法,一种是用 if-else 语句来实现,另一种是用 switch 语句来实现。

5.1 任务5:用 if 语句实现菜单的选择执行设计

只用顺序结构调用菜单的方法在实际应用中是很少见的。因为,这种结构的程序控制权不在用户手中,而在程序的开发人员手中。对于一个实用程序,菜单的控制权应当属于用户。

C 语言是用分支结构来实现对菜单的选择执行的。分支结构有两种实现方法,本任

务首先用 if-else 语句来实现菜单的选择执行。

由于任务的需要,将任务 4 原来的菜单函数稍作修改。以主菜单为例,将 MainMenu 函数中的变量定义行和倒数 3 行全部删除,只保留子菜单显示语句,其他子菜单函数也一样操作,再重新书写主函数。

```
void main() //主函数
{
    int number; //定义变量
    StuCover(); //调用封面函数
    getch();
    MainMenu(); //调用主菜单函数
    printf("\t\t 请选择序号:"); //此处加上提示
    scanf("%d",&number); //此处加上输入项
    if(number==1)
        printf("打开文件!\n"); //打印一句话
    else
        if(number==2)
            printf("保存文件!\n");//打印一句话
        else
            if(number==3)
                EditMenu(); //调用编辑子菜单函数
            else
                if(number==4)
                    DispMenu(); //调用查看子菜单函数
                else
                    if(number==5)
                        CompMenu(); //调用计算子菜单函数
                    else
                        if(number==6)
                            printf("程序说明!\n");//打印一句话
                        else
                            if(number==0)
                                printf("退出程序!\n");
                                //打印一句话
                            else
                                printf("输入错!\n");
                                //打印一句话
            }
}
```

该项目实现如果输入 0~6 之间的整型数字,将在屏幕上打印一句话或显示相应的子菜单,由于排序子菜单是属于查看子菜单下面的子菜单,它属于三级子菜单,暂时没有

显示。

单元能力训练任务分别如下。

任务 A: 正确运用 if 和 if-else 语句实现单分支和多分支结构程序。

任务 B: 正确运用条件语句实现分支程序。

任务 C: 模仿任务 5 用 if-else 语句实现“学生通讯录”的主、子菜单的选择执行。

5.2 任务 6: 用 switch 语句实现菜单的选择执行设计

实现菜单的选择执行还可以用更为简洁的 switch 语句来实现。只要将任务 5 的主函数稍加修改就能实现用 switch 语句实现菜单的选择执行。

```
void main( )                //主函数
{
    int number;              //定义变量
    StuCover( );             //调用封面函数
    getch( );
    MainMenu( );             //调用主菜单函数
    printf("\t\t 请选择序号:"); //此处加上提示
    scanf("%d",&number);     //此处加上输入项
    switch(number)
    {
        case 1:printf("打开文件!\n");break;
        case 2:printf("保存文件!\n");break;
        case 3:EditMenu();break;
        case 4:DispMenu();break;
        case 5:CompMenu();break;
        case 6:printf("程序说明!\n");break;
        case 0:printf("退出程序!\n");break;
        default:printf("输入错!\n");
    }
}
```

单元能力训练任务分别如下。

任务 A: 正确运用 switch 语句实现多分支程序。

任务 B: 模仿任务 6 用 switch 语句实现“学生通讯录”的主、子菜单的选择执行。

5.3 必备知识与理论

5.3.1 if 语句和条件运算

1. if 语句

分支结构可以改变程序的执行顺序。分支结构还可能造成某些语句不被执行,分支结构也使计算机具备了逻辑判断能力。

【例 5.1】 输入任意三个整数 num1、num2、num3,求三个数中的最大值。

```
1 #include <stdio.h>
2 void main( )
3 {
4     int num1,num2,num3,max;
5     printf("Please input three numbers:");
6     scanf("%d,%d,%d",&num1,&num2,&num3); //注意输入格式
7     if(num1>num2)
8         max=num1;
9     else
10        max=num2;
11    if(num3>max)
12        max=num3;
13    printf("max=%d\n",max);
14 }
```

程序运行结果:

```
Please input three numbers:11,22,18<回车>
max=22
```

本例中的第 1 个 if 语句,可优化为如下不带 else 子句的形式:

```
max=num1;
if(num2>max) max=num2;
```

这种优化形式的基本思想是:首先取一个数预置为 max(最大值),然后再用 max 依次与其余的数逐个比较,如果发现比 max 大的,就用它给 max 重新赋值,比较完所有的数后,max 中的数就是最大值。这种方法,对从 3 个或 3 个以上的数中找最大值的处理,非常有效。请读者仔细体会。

1) if 语句格式一

不带 else 子句的 if 语句,格式:

```
if(表达式)
    {语句组 1}
```

例如:if (x > y)

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

格式一流程图如图 5.1 所示。

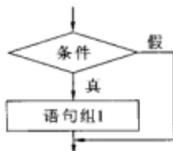


图 5.1 格式一流程图

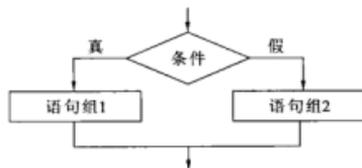


图 5.2 格式二流程图

2) if 语句格式二

带 else 子句的 if 语句,格式:

```
if(表达式)
    (语句组 1)
[else
    (语句组 2)]
```

例如:if(x>y)

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

```
else
```

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

格式二流程图如图 5.2 所示。

3) if 语句的执行过程

(1) 缺省 else 子句时。当“表达式”的值不等于 0(即判定为“逻辑真”)时,则执行语句组 1,否则直接转向执行下一条语句。如格式一流程图所示。

(2) 指定 else 子句时。当“表达式”的值不等于 0(即判定为“逻辑真”)时,则执行语句组 1,然后转向下一条语句;否则,执行语句组 2。如格式二流程图所示。

4) if 语句的嵌套与嵌套匹配原则

if 语句允许嵌套。所谓 if 语句的嵌套是指,在“语句组 1”或“语句组 2”中,又包含有 if 语句。

例如,要找出 a、b、c 中最大的数,其算法可以是:每两两进行比较,找出最大的那一个,其流程图如图 5.3 所示。

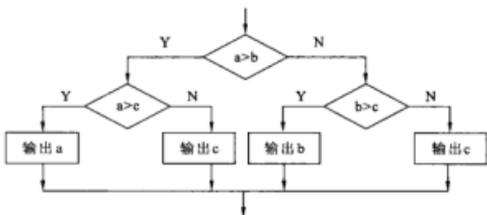


图 5.3 if 语句的嵌套流程图

程序段如下:

```

if (a>b)
    if (a>c)
        printf("%d",a);
    else
        printf("%d",c);
else
    if (b>c)
        printf("%d",b);
    else
        printf("%d",c);
  
```

由此可以看出,嵌套既可以发生在语句组 1 的位置,也可以发生在语句组 2 的位置。C 语言推荐的嵌套方式是,嵌套一律发生在 else 子句中。

【例 5.2】 从键盘上输入一个百分制成绩 score,按下列原则输出其等级:score ≥ 90 ,等级为 A;80 \leq score < 90 ,等级为 B;70 \leq score < 80 ,等级为 C;60 \leq score < 70 ,等级为 D;score < 60 ,等级为 E。

流程图如图 5.4 所示。

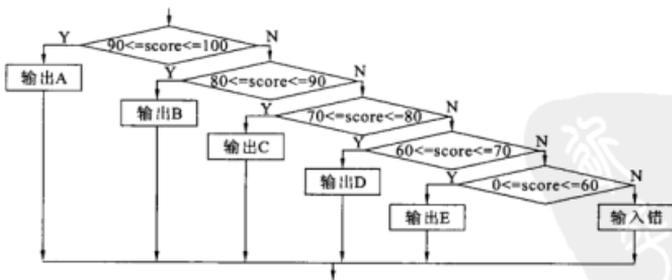


图 5.4 成绩转换成等级流程图

```

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

```
3 {
4     int score;
5     printf("请输入学习成绩:");
6     scanf("%d",&score);
7     if(score>=90&&score<=100)
8         printf("A\n");
9     else if (score>=80&&score<90)
10        printf("B\n");
11        else if (score>=70&&score<80)
12            printf("C\n");
13            else if (score>=60&&score<70)
14                printf("D\n");
15                else if (score>=0&&score<60)
16                    printf("E\n");
17                    else
18                        printf("输入错!\n");
19 }
```

程序运行结果:

请输入学习成绩:92<回车>

A

if语句嵌套时,else子句与if的匹配原则:与在它上面、距它最近、且尚未匹配的if配对。为明确匹配关系,避免匹配错误,也可以将内嵌的if语句,一律用大括号括起来。

【例 5.3】编写一个程序,从键盘上输入一个年份 year(4 位十进制数),判断其是否为闰年。闰年的条件是:能被 4 整除,但不能被 100 整除,或者能被 400 整除。

算法设计要点:

① 如果 x 能被 y 整除,则余数为 0,即如果 $x\%y$ 的值等于 0,则表示 x 能被 y 整除;

② 首先将是否闰年的标志 leap 预置为 0(非闰年),这样仅当 year 为闰年时将 leap 置为 1 即可。这种处理两种状态值的方法,对优化算法和提高程序的可读性非常有效,请读者仔细体会。

```
1 #include <stdio.h>
2 void main()
3 {
4     int year,leap=0; /* leap=0 预置为非闰年 */
5     printf("Please input the year:");
6     scanf("%d",&year);
7     if(year %4==0)
8     {
9         if(year %100!=0)
10            leap=1;
```



```
11 }
12 if(year %400==0)
13     leap=1;
14 if(leap)
15     printf("%d is a leap year.\n",year);
16 else
17     printf("%d is not a leap year.\n",year);
18 }
```

程序运行结果:

```
Please input the year:2008<回车>
2008 is a leap year.
```

利用逻辑运算能描述复杂条件的特点,可将上述程序优化如下。

```
1 #include <stdio.h>
2 void main( )
3 {
4     int year;//,leap=0;    /* leap=0 预置为非闰年*/
5     printf("Please input the year:");
6     scanf("%d",&year);
7     if(year%4==0&&year%100!=0||year%400==0)
8         printf("%d is a leap year.\n",year);
9     else
10         printf("%d is not a leap year.\n",year);
11 }
```

程序运行结果:

```
Please input the year:2008<回车>
2008 is a leap year.
```

5) 说明

(1) if 语句中“表达式”必须用“(”和“)”括起来,它的值为逻辑值。除常见的关系表达式或逻辑表达式外,也允许是其他类型的数据,如整型、实型、字符型等。

(2) else 子句(可选)是 if 语句的一部分,必须与 if 配对使用,不能单独使用。

(3) 当 if 和 else 下面的语句组仅由一条语句构成时,也可不使用复合语句形式(即去掉大括号)。

(4) if 语句允许嵌套,但嵌套的层数不宜太多。在实际编程时,应当适当控制嵌套的层数(2~3层)。

注意:不管是简单语句,还是复合语句中的每条语句,结束的分号必不可少,如果在 if (表达式)后面加分号,分号就是语句组 1。

2. 条件运算

1) 格式

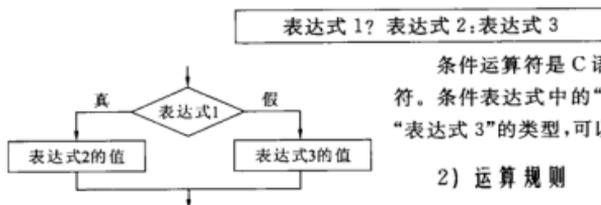


图 5.5 条件运算表达式流程图

条件运算符是 C 语言中唯一一个三目运算符。条件表达式中的“表达式 1”、“表达式 2”、“表达式 3”的类型,可以各不相同。

2) 运算规则

如果“表达式 1”的值为非 0(即逻辑真),则运算结果等于“表达式 2”的值;否则,运算结果

等于“表达式 3”的值,如图 5.5 所示。

从图 5.5 流程图可以清楚地看出,条件运算是一种简化了的 if-else 表达式。

3) 条件运算符的优先级与结合性

条件运算符的优先级,高于赋值运算符,但低于关系运算符和算术运算符。其结合性为“自右至左”(即右结合性)。

【例 5.4】 从键盘上输入一个字符,如果它是大写字母,则把它转换成小写字母输出,否则,直接输出。

```

1 #include <stdio.h>
2 void main()
3 {
4     char ch;
5     printf("Input a character: ");
6     scanf("%c",&ch);
7     ch=(ch>='A' && ch<='Z')?(ch+32):ch;
8     printf("ch=%c\n",ch);
9 }
  
```

程序运行结果:

```

Input a character: B<回车>
ch=b
  
```

5.3.2 switch 语句

if 语句可以处理多分支,但分支不宜太多,因此,C 语言提供了 switch 语句用于直接处理多分支选择。

【例 5.5】 用 switch 语句重做例 5.2。

```

1 #include <stdio.h>
2 void main( )
3 {
4     int score, grade;
5     printf("Input a score (0~100): ");
6     scanf("%d", &score);
7     grade=score/10; /*将成绩整除 10,转化成 switch 语句中的 case 标号*/
8     switch (grade)
9     {
10         case 10:
11         case 9: printf("grade=A\n"); break;
12         case 8: printf("grade=B\n"); break;
13         case 7: printf("grade=C\n"); break;
14         case 6: printf("grade=D\n"); break;
15         case 5:
16         case 4:
17         case 3:
18         case 2:
19         case 1:
20         case 0: printf("grade=E\n"); break;
21         default: printf("error!\n");
22     }
23 }

```

程序运行结果：

```

Input a score (0~100): 90<回车>
grade=A

```

流程图如图 5.6 所示。

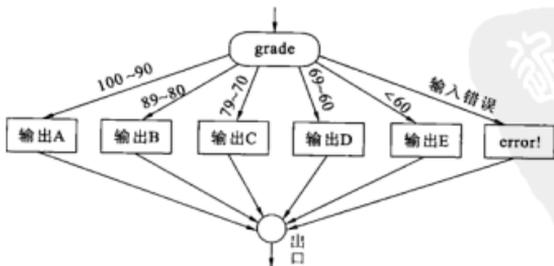


图 5.6 用 switch 语句实现输入分数输出等级流程图

1. switch 语句格式

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

2. Switch 语句的执行过程

(1) 当 switch 后面“表达式”的值,与某个 case 后面的“常量表达式”的值相同时,就执行该 case 后面的语句(组);当执行到 break 语句时,跳出 switch 语句,转向执行 switch 语句的下一条。

(2) 如果没有任何一个 case 后面的“常量表达式”的值,与“表达式”的值匹配,则执行 default 后面的语句(组)。然后,再执行 switch 语句之后的下一条。

3. 说明

(1) switch 后面的“表达式”,可以是 int、char 或枚举型中的一种。

(2) 每个 case 后面“常量表达式”的值,必须各不相同,否则会出现相互矛盾的现象(即对表达式的同一值,有两种或两种以上的执行方案)。

(3) case 后面的常量表达式只起语句标号作用,并不进行条件判断。系统一旦找到入口标号,就从此标号开始执行,不再进行标号判断,所以必须加上 break 语句,以便结束 switch 语句。

想一想:如果去掉例 5.5 程序中的所有 break 语句,且输入的成绩为 75,输出结果会如何?

(4) 各 case 及 default 子句的先后次序,不影响程序的执行结果。

(5) 多个 case 子句,可共用同一语句(组)。例如,在例 5.5 中的“case 10:”和“case 9:”共用语句“printf(“grade=A\n”); break;”,“case 5:”~“case 0:”共用语句“printf(“grade=E\n”); break;”。

(6) 用 switch 语句实现的多分支结构程序,完全可以用 if 语句或 if 语句的嵌套来实现。

【例 5.6】 已知某公司员工的保底薪水为 500 元,某月所接工程的利润 profit(整数)与利润提成关系如下(计量单位:元):

profit ≤ 1000	没有提成;
1000 < profit ≤ 2000	提成 10%;
2000 < profit ≤ 5000	提成 15%;
5000 < profit ≤ 10000	提成 20%;

$10000 < \text{profit}$ 提成 25%。

算法设计要点:为使用 switch 语句,必须将利润 profit 与提成的关系转换成某些整数与提成的关系。分析本题可知,提成的变化点都是 1000 的整数倍(1000、2000、5000...),如果将利润 profit 整除 1000,则当:

$\text{profit} \leq 1000$	对应 0,1
$1000 < \text{profit} \leq 2000$	对应 1,2
$2000 < \text{profit} \leq 5000$	对应 2,3,4,5
$5000 < \text{profit} \leq 10000$	对应 5,6,7,8,9,10
$10000 < \text{profit}$	对应 10,11,12...

为解决相邻两个区间的重叠问题,最简单的方法就是,利润 profit 先减 1(最小增量),然后再整除 1000 即可:

$\text{profit} \leq 1000$	对应 0
$1000 < \text{profit} \leq 2000$	对应 1
$2000 < \text{profit} \leq 5000$	对应 2,3,4
$5000 < \text{profit} \leq 10000$	对应 5,6,7,8,9
$10000 < \text{profit}$	对应 10,11,12...

```

1 #include <stdio.h>
2 void main()
3 {
4     long profit;
5     int grade;
6     float salary=500;
7     printf("Input profit: ");
8     scanf("%ld", &profit);
9     grade= (profit-1)/1000;
10    switch(grade)
11    {
12        case 0: break; /*profit≤1000*/
13        case 1: salary+=profit*0.1; break; /*1000<profit≤2000*/
14        case 2:
15        case 3:
16        case 4: salary+=profit*0.15; break; /*2000<profit≤5000*/
17        case 5:
18        case 6:
19        case 7:
20        case 8:
21        case 9: salary+=profit*0.2; break; /*5000<profit≤10000*/
22        default: salary+=profit*0.25; /*10000<profit*/
23    }

```

```

24     printf("salary=%.2f\n", salary);
25 }

```

程序运行结果:

```

Input profit:8500<回车>
salary=2200.00

```

5.4 常见错误及处理方法

```

常见错误 1: if(x>y);
           printf("%d",x);
           else
           printf("%d",y);

```

在 if(表达式)后不正确的使用分号,编译时系统将提示 if 与 else 匹配非法的错误 (“illegal else without matching if”),这样就得不到预期的结果。这是因为,if(表达式)后如果有分号,这个分号就是 if 的执行语句而不是下面的输出语句,所以得不到预期的结果。

常见错误 2:将相等关系判断语句如 if(x=0)误写成 if(x=0),这样表达式的结果永远就是“假”,而不可能是“真”。

常见错误 3:不清楚 if(变量)与 if(变量==1)等价还是与 if(变量!=0)等价。根据 C 语言对逻辑值概念的扩展,if(变量)应当与 if(变量!=0)等价。这是因为,当变量为非零值时,C 语言将其定义为逻辑真,而“变量!=0”表达式的值也为真;当变量为零值时,C 语言将其定义为逻辑假,而“变量!=0”表达式的值也为假。

常见错误 4:switch 语句中的 case 子句中不使用 break 语句。如果在 switch 语句中不使用 break 语句,switch 语句获得入口之后,就会一直执行到 switch 语句的结束,有时这是不符合实际的,所以,在编程实践中,除最后一条子句之外,其他的子句在结束之前都应加上 break 语句。

5.5 深入训练

(1) 输入一个整数,如果这个整数能被 3 整除,则输出该整数,否则输出这个整数的平方。

(2) 编写一个程序,从键盘上输入四个整数,输出其中的最小值。

(3) 输入四个整数,按从小到大的顺序输出。

(4) 输入某学生的成绩,输出相应的信息。成绩在 90~100 分之间,输出“Very good!”,成绩在 70~89 分之间,输出“Good!”,成绩在 60~69 分之间,输出“Pass!”,60 分以下输出“No pass!”。(分别用 if-else 语句和 switch 语句实现)

(5) 编程实现输入三角形的三边长度,计算三角形面积。要求:首先判断输入的三条边能否围成三角形,如果能围成三角形就计算并输出三角形面积;否则,输出这三条边不

能围成三角形。

另外,将这个程序修改为在输出三角形面积的同时,还能判断这个三角形是否是等边、等腰、直角或一般三角形?

习 题 5

5.1 填空题

(1) 如果执行 `if(x>3)printf("A");else printf("B");` 后屏幕上显示的是 B,说明 $x>3$ 的值是_____。

(2) 从两个数中挑选出最大的至少需要进行_____次比较,从三个数中挑选出最大的至少需要进行_____次比较,从 n 个数中挑选出最大的至少需要进行_____次比较。

(3) 通过执行 `x=y,y=z,z=x;` 可以交换变量_____和变量_____的值。

(4) 执行 `if(x||!x)printf("abc");else printf("xyz");` 后屏幕上显示的是_____。

(5) 与

```
switch(k)
{
    case 1:
    case 2:printf("A");break;
    case 3:
    case 4:
    case 5: printf("B");break;
    default: printf("C");
}
```

等价的 if 语句是_____。

(6) 与

```
if(x>0&&x<4)printf("X");
else if(x>18&&x<=22)printf("Y");
else printf("Z");
```

(其中 x 是 int 型变量)等价的 switch 语句是_____。

(7) 若 $a=10, b=20$, 则表达式 $a>b?a+a+b$ 的值是_____。

(8) 若 $a=10, b=20, c=30, d=40$, 则表达式 $a>b?a:c>d?c:d$ 的值是_____。

(9) 设有 `int x=0,y=1;` 则以下表达式的值为_____。

```
x+*x--!='y'?8-4:y
```

(10) 若有 $x=1, y=2, z=3$, 则表达式 $(x<y?x,y)==z++$ 的值是_____。

5.2 判断题(判断下列叙述的正确性,正确的请打“√”,错误的请打“×”)

(1) C 语言规定,else 子句总是与它上面的最近的 if 配对。 ()

(2) 在 switch 语句中必须使用 break 语句。 ()

(7) 若有条件表达式 $(exp)?a++ : b--$, 则以下表达式中能完全等价于表达式 (exp) 的是()。

- A. $(exp==0)$ B. $(exp!=0)$ C. $(exp==1)$ D. $(exp!=1)$

(8) 以下程序的运行结果是()。

```
main()
{
    int k=4,a=3,b=2,c=1;
    printf("\n%d\n",k<a?k:c<b?c:a);
}
```

- A. 4 B. 3 C. 2 D. 1

(9) 若运行时给变量 x 输入 12, 则以下程序的运行结果是()。

```
main()
{
    int x, y;
    scanf("%d", &x);
    y=x>12?x+10:x-12;
    printf("%d\n", y);
}
```

- A. 0 B. 22 C. 1 D. 2



单元6 项目菜单的循环选择执行设计

能力目标

- 掌握用 for 语句、while 语句和 do-while 语句实现循环结构的方法。
- 掌握 break 语句和 continue 语句的使用方法和区别。
- 掌握用 for 循环、while 循环和 do-while 循环实现二、三重循环的方法。
- 掌握 if 语句和 switch 语句与循环结构的混合使用方法。
- 能用循环语句实现“班级学生成绩管理系统”中的主菜单和子菜单的循环操作。

知识目标

- 理解 for 语句、while 语句、do-while 语句的定义格式和执行过程。
- 理解循环三要素。
- 理解循环的嵌套定义原则与方法。
- 理解 break 语句和 continue 语句在循环结构中的不同意义。

学习提示

循环结构是一种重要并且较难掌握的程序结构。学习时要充分理解实现循环的机制,掌握循环三要素在循环结构中的作用,循环嵌套的实现方法。

另外,通过学习养成良好的编程习惯是进一步学好 C 语言的必要技能,因此在学习实践中要注意良好编程习惯的训练。

6.1 任务7:用循环语句实现项目主菜单的选择执行设计

任务5和任务6实现了菜单的选择执行,虽然实现了用户选择执行菜单功能,但每执行一个菜单后,程序就结束了,这还是不能满足用户的需要。用户往往需要程序在没有结束之前都能被操作。要实现上述功能,必须使用循环结构。

由于实现循环选择执行项目菜单是一种比较复杂的结构,为了便于学习,首先用三种方法实现主菜单的循环选择执行,即主菜单实现循环选择,而子菜单暂时不实现循环选择。

1. 用 for 循环实现已知循环次数的循环

该循环规定了循环执行的次数,当循环达到所规定的次数后将会退出循环。

```
void main()
{
    int i;                //定义循环变量
    int choose;
    system("cls");
    StuCover();
    getch();
    for(i=0;i<8;i++)
    {
        MainMenu();      //调用主菜单函数
        printf("\t\t请输入序号:");
        scanf("%d",&choose);
        switch(choose)//主菜单的 switch
        {
            case 1:printf("打开文件!\n");getch();break;
            case 2:printf("保存文件!\n");getch();break;
            case 3:EditMenu();getch();break;
            case 4:DispMenu();getch();break;
            case 5:CompMenu();getch();break;
            case 6:printf("程序说明!\n");getch();break;
            case 0:printf("退出程序!\n");getch();break;
            default:printf("输入错!\n");getch();
        }
    }
}
```

2. 用 while 循环实现未知循环次数的循环

```
void main()
{
    int choose;
    system("cls");
    StuCover();
    getch();
    MainMenu();//调用主菜单函数
    printf("\t\t请输入序号:");
    scanf("%d",&choose);
```



```
while(choose!=0)
{
    switch(choose)//主菜单的 switch
    {
        case 1:printf("打开文件!\n");getch();break;
        case 2:printf("保存文件!\n");getch();break;
        case 3:EditMenu();getch();break;
        case 4:DispMenu();getch();break;
        case 5:CompMenu();getch();break;
        case 6:printf("程序说明!\n");getch();break;
        case 0:printf("退出程序!\n");getch();break;
        default:printf("输入错!\n");getch();
    }
    MainMenu(); //调用主菜单函数
    printf("\t\t请输入序号:");
    scanf("%d",&choose);
}
}
```

该循环只有输入了 0 才会退出循环,但不能显示“退出程序!”,即当输入 0 后,循环再不执行,从而退出循环。

3. 用 do-while 循环实现未知循环次数的循环

```
void main()
{
    int choose;
    system("cls");
    StuCover();
    getch();
    do
    {
        MainMenu(); //调用主菜单函数
        printf("\t\t请输入序号:");
        scanf("%d",&choose);
        switch(choose) //主菜单的 switch
        {
            case 1:printf("打开文件!\n");getch();break;
            case 2:printf("保存文件!\n");getch();break;
            case 3:EditMenu();getch();break;
            case 4:DispMenu();getch();break;
```



```
        printf("\n\n\n\n\n\n\t\t\t 操作结束,返回上级菜单!");
        getch();
        system("cls");
    }
}

void main()
{
    int choose,editnum,dispnum,compnum,sortnum;//定义 5 个输入变量
    system("cls");
    StuCover();
    getch();
    while(1)//外循环
    {
        MainMenu();//调用主菜单函数
        printf("\t\t 请输入序号:");
        scanf("%d",&choose);
        switch(choose)//主菜单的 switch
        {
            case 1:printf("打开文件!\n");getch();break;
            case 2:printf("保存文件!\n");getch();break;
            case 3:
                do//内循环 1
                {
                    EditMenu();//调用编辑子菜单函数
                    printf("\t\t 请输入序号:");
                    scanf("%d",&editnum);
                    switch(editnum) //编辑子菜单 switch
                    {
                        case 1:printf("增加记录!\n");getch();break;
                        case 2:printf("删除记录!\n");getch();break;
                        case 3:printf("修改记录!\n");getch();break;
                        case 0:Quit(0);break;
                    } //编辑子菜单 switch 结束
                }while(editnum!=0); //内循环 1 结束
            break;
            case 4:
                do //内循环 2
                {
                    DispMenu(); //调用查看子菜单函数
```

```
printf("\t\t请输入序号:");
scanf("%d",&dispnum);
switch(dispnum) //显示子菜单 switch
{
case 1:printf("查看选定记录!\n");getch();break;
case 2:printf("显示全部记录!\n");getch();break;
case 3:
do //内循环 3
{
SortMenu(); //调用排序子菜单函数
printf("\t\t请输入序号:");
scanf("%d",&sortnum);
switch(sortnum)//排序子菜单 switch
{
case 1:printf("按升序排序!\n");getch();break;
case 2:printf("按降序排序!\n");getch();break;
case 0:Quit(0);break;
} //排序子菜单 switch 结束
}while(sortnum!=0); //内循环 3 结束
break;
case 4:printf("显示不及格记录!\n");getch();break;
case '0':Quit(0);break;
} //显示子菜单 switch 结束
}while(dispnum!=0); //内循环 2 结束
break;
case 5:
do //内循环 4
{
CompMenu(); //调用计算子菜单函数
printf("\t\t请输入序号:");
scanf("%d",&compnum);
switch(compnum) //计算子菜单 switch
{
case 1:printf("计算总成绩和平均成绩!\n");getch();break;
case 2:printf("计算最高分!\n");getch();break;
case 3:printf("计算最低分!\n");getch();break;
case 0:Quit(0);break;
} //计算子菜单 switch 结束
}while(compnum!=0); //内循环 4 结束
```

```

        break;
    case 6:printf("程序说明!\n");getch();break;//程序说明
    case 0:Quit(1);break;
    }//主菜单的 switch 结束
} //外循环结束
}

```

本任务涉及的其他内容与前面的任务相同,这里不再冗述。

单元能力训练任务分别如下。

任务 A:能运用 for、while 和 do-while 嵌套循环输出乘法口诀表。

任务 B:使用 break 语句和 continue 语句实现对循环的控制。

任务 C:使用 for、while 和 do-while 循环嵌套实现“班级学生成绩管理系统”的主、子菜单的循环选择执行。

任务 D:模仿任务 8 实现“学生通讯录”的主、子菜单的循环选择执行。

6.3 必备知识与理论

目前计算机是不能进行“创造性”劳动的,但它却是“重复性”劳动的高手,而重复性劳动常常可以通过循环来实现。

求 1~100 的累计和。根据已有的知识,可以用“ $1+2+\dots+100$ ”来求解,但显然很烦琐。现在换个思路来考虑:首先设置一个变量 sum,其初值为 0,用来存放计算之和。利用 $sum+=n$ 来计算(n 依次取 1、2、……、100),只要解决以下三个问题即可:

- ① 将 n 的初值置为 1;
- ② 每执行 1 次“ $sum+=n$ ”后, n 增 1;
- ③ 当 n 增到 101 时,停止计算。此时, sum 的值就是 1~100 的累计和。

可以用 if 语句+goto 语句(无条件转向语句)来实现上述计算。

goto 语句的一般格式:

goto 语句标号;

其功能为:使系统转向标号所在的语句行执行。

语句标号用标识符表示,它的命名规则与变量名的命名规则相同。例如:

```

goto label_1; /*合法*/
goto 123;    /*不合法*/

```

用 goto 语句和 if 语句构成循环。使用 goto 语句实现求解 1~100 累计和的程序如下(流程图如图 6.1 所示):

```

void main( )
{
    int n=1,sum=0;
loop: sum+=n;n++;
}

```

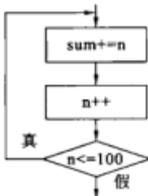


图 6.1 用 if 和 goto 语句形成的循环流程图

```

    if(n<=100) goto loop;
    printf("sum=%d\n", sum);
}

```

注意:结构化程序设计方法限制使用 goto 语句。因为滥用 goto 语句,将破坏结构化程序设计的结构,导致程序结构无规律、可读性差。

由于需要经常使用这种重复计算结构(称为循环结构),C 语言提供了三种循环语句来实现,以简化并规范循环结构的程序设计,即 for 语句、while 语句和 do-while 语句。

6.3.1 for 语句

在三种循环语句中,for 语句最为灵活,不仅可用于循环次数已经确定的情况,也可用于循环次数虽不确定,但给出了循环继续(结束)条件的循环。

【例 6.1】 求 1~100 的累加和。

```

1 #include <stdio.h>
2 void main( )
3 {
4     int i,sum=0;    /*将累加器 sum 初始化为 0*/
5     for(i=1; i<=100; i++)
6         sum+=i;    /*实现累加*/
7     printf("sum=%d\n",sum);
8 }

```

程序运行结果:

sum=5050

【例 6.2】 求 n 的阶乘 $n!(n!=1*2*\dots*n)$ 。

```

1 #include <stdio.h>
2 void main( )
3 {
4     int i, n;
5     long fact=1;    /*将累乘器 fact 初始化为 1*/
6     printf("Input n: ");
7     scanf("%d", &n);
8     for(i=1; i<=n; i++)
9         fact*=i;    /*实现累乘*/
10    printf("%d !=%ld\n", n, fact);
11 }

```

程序运行结果:

Input n: 5<回车>

5 !=120

1. for 语句格式

```
for([循环变量赋初值];[循环继续条件];[循环变量改变])
{
    循环体语句组;
}
```

2. for 语句的执行过程

执行过程流程图如图 6.2 所示。

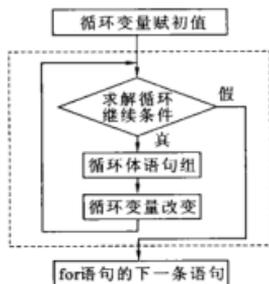


图 6.2 for 语句的执行过程流程图

- (1) 求解“循环变量赋初值”表达式。
- (2) 求解“循环继续条件”表达式。如果其值非 0，执行(3)；否则，执行(4)。
- (3) 执行循环体语句组，并求解“循环变量改变”表达式，然后转向(2)。
- (4) 执行 for 语句的下一条语句。

3. 说明

- (1) “循环变量赋初值”、“循环继续条件”和“循环变量改变”部分均可缺省，甚至全部缺省，但其间的分号不能省略(如 for(;;))。

① 循环变量赋初值省略。

例如：i=1;

```
for (; i<=100; i++)
    sun+=i;
```

由此可以看出，循环变量赋初值并不是真的省略了，而是换了一个地方。

② 循环变量改变也可以省略。

例如：i=1;

```
for (; i<=100;)
{
    sun+=i;
    i++;
}
```

从上例可以看出，循环变量改变并没有省略，而是换了一个地方。如果一定要省略循环变量改变这个表达式，将不能使循环条件达到“假”，因而不能结束循环，该循环将成为死循环，死循环一般是没有意义的，应当尽量避免。

③ 循环继续条件省略。

例如：i=1;

```
for (; ;)
```



```

{
    sum+=i;
    i++;
    if(i>=101)
        break;
}

```

如果循环条件省略,就不判断循环条件,流程图如图 6.3 所示,即认为循环条件始终为真,循环无终止地进行,如果没有别的办法退出循环,将成为死循环,因此,这时应另外设法保证循环能正常结束。

(2) 当循环体语句组仅由一条语句构成时,可以不使用复合语句形式。

(3) “循环变量赋初值”表达式,既可以是给循环变量赋初值的赋值表达式,也可以是其他表达式(如逗号表达式)。

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

(4) “循环继续条件”部分是一个逻辑量,除一般的关系(或逻辑)表达式外,也允许是数值(或字符)表达式。C 语言将非 0 值看成是逻辑真,将 0 值看成是逻辑假。

6.3.2 while 语句

1. while 语句格式

while 又称为“当型”循环语句,它的格式为:

```

while(循环继续条件)
{
    循环体语句组;
}

```

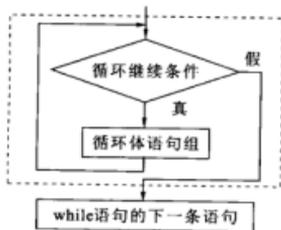


图 6.4 while 语句执行过程流程图



图 6.3 省略了循环条件的 for 循环流程图

2. while 语句的执行过程

执行过程流程图如图 6.4 所示。

(1) 求解“循环继续条件”表达式。如果其值为非 0,转(2);否则转(3)。

(2) 执行循环体语句组,然后转(1)。

(3) 执行 while 语句的下一条。

显然,while 循环是 for 循环的一种简化形式(缺省“变量赋初值”和“循环变量改变”表达式)。

while 循环的特点是,先判断条件后再执行循环体。

【例 6.3】 用 while 语句求 1~100 的累计和。

```

1 #include <stdio.h>
2 void main( )
3 {
4     int i=1,sum=0;    /*初始化循环控制变量 i 和累加器 sum*/
5     while( i<=100 )
6     {
7         sum+=i;      /*实现累加*/
8         i++;        /*循环控制变量 i 增 1*/
9     }
10    printf("sum=%d\n",sum);
11 }

```

程序运行结果：

sum=5050

6.3.3 do-while 语句

1. do-while 语句格式

do-while 语句又称为“直到型”循环语句，它的格式为：

```

do
{
    循环体语句组；
}while(循环继续条件)；

```

注意：尾行的分号不能省略。

当循环体语句组仅由一条语句构成时，可以不使用复合语句形式。

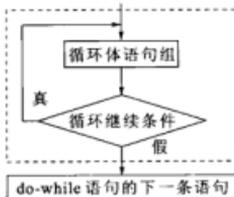


图 6.5 do-while 语句执行过程流程图

2. do-while 语句执行过程

- (1) 执行循环体语句组。
- (2) 计算“循环继续条件”表达式。如果“循环继续条件”表达式的值为非 0(真)，则转向(1)继续执行；否则，转向(3)。

(3) 执行 do-while 的下一条语句。

执行过程流程图如图 6.5 所示。

do-while 循环语句的特点是：先执行循环体语句组，

然后再判断循环条件。

【例 6.4】 用 do-while 语句求解 1~100 的累计和。

```

1 #include <stdio.h>
2 void main( )

```

```

3 {
4     int i=1, sum=0;    /*定义并初始化循环控制变量 i 和累加器*/
5     do
6     {
7         sum+=i;      /*累加*/
8         i++;
9     }while(i<=100); /*循环继续条件:i<=100*/
10    printf("sum=%d\n",sum);
11 }

```

程序运行结果:

```
sum=5050
```

do-while 语句比较适用于处理:不论条件是否成立,先执行 1 次循环体语句组的情况。除此之外,do-while 语句能实现的,for 语句也能实现,而且更简洁。

想一想:如果例 6.3 和例 6.4 的循环条件一开始就为假,它们的运算结果如何?读者可以试一试。

6.3.4 循环的嵌套

如果循环体内又包含另一个完整的循环结构,则称为循环的嵌套。循环可以允许多层嵌套。

for 语句、while 语句、do-while 语句都允许进行嵌套,并且可以进行相互嵌套。

1. for 循环嵌套

for 循环嵌套如图 6.6~图 6.8 所示。

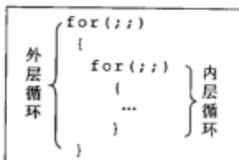


图 6.6 for 循环与 for 循环的嵌套

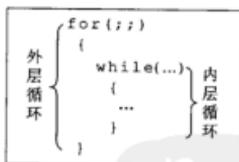


图 6.7 for 循环与 while 循环的嵌套

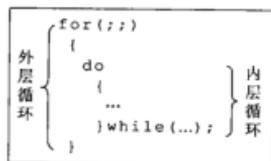


图 6.8 for 循环与 do-while 循环的嵌套

2. while 循环嵌套

while 循环嵌套如图 6.9~图 6.11 所示。

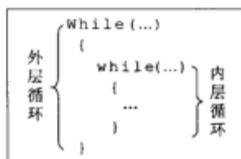


图 6.9 while 循环与 while 循环的嵌套

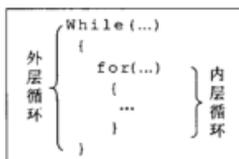


图 6.10 while 循环与 for 循环的嵌套

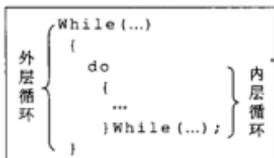


图 6.11 while 循环与 do-while 循环的嵌套

3. do-while 循环嵌套

do-while 循环嵌套如图 6.12~图 6.14 所示。

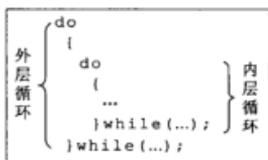


图 6.12 do-while 循环与 do-while 循环的嵌套

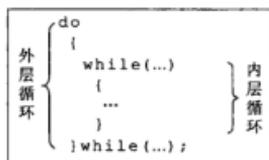


图 6.13 do-while 循环与 while 循环的嵌套

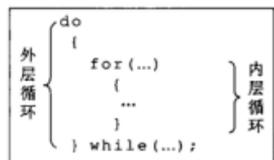


图 6.14 do-while 循环与 for 循环的嵌套

4. 关于循环的几点注意

(1) 四种循环都可以处理同一问题,没有孰优孰劣之分。但一般不用 goto 语句来实现循环。

(2) while 和 do-while 循环,只在 while 后面指定循环条件,因此在循环体中要有使循环趋于结束的语句。

(3) 一般情况下要实现循环,不要忘记“循环三要素”,即“循环变量赋初值”、“循环结束条件(或继续条件)”、“循环变量的改变”(可增可减)。

6.3.5 break 语句与 continue 语句

为了使循环控制更加灵活,C语言提供了 break 语句和 continue 语句。

1. break 语句与 continue 语句格式

break;
continue;

2. 功能

在前面已经知道 break 语句可以跳出 switch 结构,实际上 break 语句还可以用于循环语句中。break 语句和 continue 语句常常和 if 语句配合使用,达到控制循环的目的。

(1) break: 强行结束循环,转向执行循环语句的下一条语句。

(2) continue: 结束本次循环。对于 for 循环,跳过循环体其余语句,转向循环变量改变表达式的计算;对于 while 和 do-while 循环,跳过循环体其余语句,但转向循环继续条件的判定。

(3) break 和 continue 语句对循环控制的影响如图 6.15、图 6.16 所示。

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

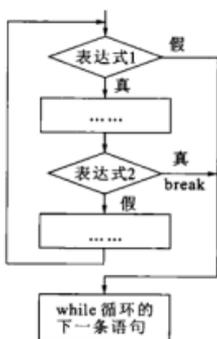


图 6.15 使用 break 语句流程图

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

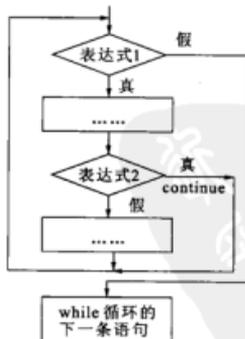


图 6.16 使用 continue 语句流程图

3. 说明

- (1) break 语句能用于循环语句和 switch 语句中,但 continue 语句只能用于循环语句中。
- (2) 循环嵌套时,break 语句和 continue 语句只影响包含它们的最内层循环,与外层循环无关。

【例 6.5】 把 100~200 之间的能被 3 整除的数输出,并且每行输出 10 个。

```

1 #include <stdio.h>
2 void main()
3 {
4     int n,count=0;           //count 用来记数
5     for(n=100;n<=200;n++) //循环
6     {
7         if(n%3!=0)          //不能被 3 整除的数跳过
8             continue;
9         count++;
10        printf("%5d",n);
11        if(count%10==0)     //如果 count 能被 10 整除,输出回车换行
12            printf("\n");
13    }
14    printf("\n");
15 }

```

程序运行结果:

```

102 105 108 111 114 117 120 123 126 129
132 135 138 141 144 147 150 153 156 159
162 165 168 171 174 177 180 183 186 189
192 195 198

```

【例 6.6】 输出 10~100 之间的全部素数。所谓素数 n 是指,除 1 和 n 之外,不能被 $2\sim(n-1)$ 之间的任何整数整除。

算法设计要点如下。

- (1) 显然,只要设计出判断某数 n 是否是素数的算法,外面再套一个 for 循环即可。
- (2) 判断某数 n 是否是素数的算法:根据素数的定义,用 $2\sim(n-1)$ 之间的每一个数去整除 n ,如果都不能被整除,则表示该数是一个素数。

判断一个数是否能被另一个数整除,可通过判断它们整除后的余数是否为 0 来实现。

```

1 #include <stdio.h>
2 void main()
3 {
4     int i=11, j, counter=0;
5     for(;i<=100;i+=2)      /*外循环:为内循环提供一个整数 i*/
6     {

```

```

7         for(j=2;j<=i-1;j++) /*内循环:判断整数 i 是否是素数*/
8     {
9         if(i%j==0) /*i 不是素数*/
10        break; /*强行结束内循环,执行下面的 if 语句*/
11    }
12    if(j>=i) /*整数 i 是素数:输出,计数器加 1*/
13    {
14        printf("%6d",i);
15        counter++;
16    }
17    if(counter!=0&&counter%10==0)/*每输出 10 个数换一行*/
18    {
19        printf("\n");
20        counter=0; //记计数器重新归 0
21    }
22 }
23 printf("\n");
24 }

```

程序运行结果:

```

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

```

6.4 扩展知识与理论

6.4.1 良好的源程序书写习惯

顺序程序段中的所有语句(包括说明语句),一律与本顺序程序段的首行左对齐,并且采用缩进对齐的方式,使程序便于阅读。

必要的注释可有效地提高程序的可读性,从而提高程序的可维护性。

在 C 语言源程序中,注释可分为以下三种情况:

- ① 在函数体内对语句的注释;
- ② 在函数之前对函数的注释;
- ③ 在源程序文件开始处,对整个程序的总体说明。

加注释的原则是:如果不加注释,理解起来就会有困难,或者虽无困难但浪费时间。

函数体内的语句,是由顺序结构、分支结构和循环结构三种基本结构构成的。

1. 顺序结构

在每个顺序程序段(由若干条语句构成)之前,用注释说明其功能。除对复杂的语句

外,一般没有必要对每条语句都加以注释。

2. 分支结构

在 C 语言中,分支结构是由 if 语句或 switch 语句来实现的。一般地说,要在前面说明其作用,在每个分支语句行的后面,说明该分支的含义,如下所示。

1) if 语句

```
/*说明功能*/  
if(条件表达式)                /*条件成立时的含义*/  
    {...}  
else                            /*入口条件的含义*/  
    {...}
```

2) switch 语句

```
/*说明功能*/  
switch(表达式)  
{  
    case 常量表达式 1:        /*该入口值的含义*/  
        语句组;  
    ...  
    case 常量表达式 n:        /*该入口值的含义*/  
        语句组;  
    default:                    /*该入口值的含义*/  
        语句组;  
}
```

如果条件成立(或入口值)的含义已经很明确,也可不再加以注释。

3. 循环结构

在 C 语言中,循环结构由循环语句 for、while 和 do-while 来实现。

作为注释,应在它们的前面说明其功能,在循环条件判断语句行的后面,说明循环继续条件的含义,如下所示。

1) for 语句

```
/*功能说明*/  
for(变量初始化;循环条件;变量改变)    /*循环继续条件的含义*/  
    {...}
```

2) while 语句

```
/*功能说明*/
```

```
while(循环条件) /*循环继续条件的含义*/
{...}
```

3) do-while 语句

```
/*功能说明*/
do {...}
while(循环条件); /*循环继续条件的含义*/
```

如果有循环嵌套,还应说明每层循环各控制什么。

6.4.2 常见错误及处理方法

常见错误 1:分不清循环变量和一般变量。给出一个判断原则,能够控制循环的变量称为循环变量。

常见错误 2:在循环语句中忘记给循环变量赋初值。由于循环变量没有赋初值,系统就会给出一个随机值,而这个随机值往往是一个很大的负数,这样无法控制好循环。因此无论用什么循环语句,都不要忘记给循环变量赋初值。

常见错误 3:在循环语句中不能很好地设置循环变量的步长。如果不能很好地控制循环变量的步长,有时会使循环无法达到结束循环的条件,这样会造成死循环。

常见错误 4:循环语句中没有结束循环的条件,对这样的循环如果没有别的办法实现结束循环,该循环将是死循环。

常见错误 5:认为 for 循环中控制循环的变量一定要是整型数据,而不能是其他类型的数据。这是不对的,循环变量还可以是浮点型和字符型,如下所示。

```
#include <stdio.h>
void main()
{
    double i;
    for(i=1.1;i<2;i+=0.1)
        printf("%5.1f",i);
}
```

程序运行结果:

```
1.1 1.2 1.3 1.4 1.5 1.6 1.7 1.8 1.9
```

又如:

```
#include <stdio.h>
void main()
{
    char ch;
    for(ch='a';ch<='z';ch++)
        printf("%3c",ch);
}
```

程序运行结果:

```
a b c d e f g h i j k l m n o p q r s t u v w x y z
```



6.5 深入训练

- (1) 用 for 循环设计一程序,实现求 1~100 的奇数和。
- (2) 用 do-while 循环设计一程序,实现求 1~10 之间的累计乘。
- (3) 设计一个程序,显示输出如下所示的三角形(要求用循环语句实现)。

```

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

- (4) 输出所有的“水仙花数”,所谓水仙花数是指一个三位十进制数,该数的各位数字立方之和等于该数本身。例如,153 是一个水仙花数,因为 $1^3+5^3+3^3=153$ 。
- (5) 猴子吃桃问题。第 1 天摘下桃子若干,当即吃掉一半,又多吃一个;第 2 天将剩余部分吃掉一半还多一个;依此类推,到第 10 天只剩下一个,问第 1 天共摘了多少个桃子?
- (6) 中国古典数学问题。公鸡 5 元一只,母鸡 3 元一只,雏鸡三只 1 元。问:用 100 元钱买 100 只鸡,公鸡、母鸡、雏鸡各几只?
- (7) 从键盘上输入任意多个正整数(以数值 0 作为结束标志),统计数据个数、累计和、平均值、找出最大值和最小值。
- (8) 设计一个程序,计算 1~20 之间所有能被 3 整除的数之和。
- (9) 设计一个程序,计算 $1!+2!+3!+\dots+10!$ 。

习 题 6

6.1 填空题

- (1) 执行 `for(i=0;i<28;i++)printf(“*”);` 将输出_____个*号。
- (2) 执行 `for(i=20;i>=0;i--)printf(“*”);` 将输出_____个*号。
- (3) 与 `int i=10;while(i<100){printf(“p”);i++;}` 这两条语句等价的 for 语句是_____。
- (4) 与 `for(i=0;i<10;i++)printf(“%d”,i);` 语句等价的 while 循环是_____。
- (5) break 语句的功能是_____,continue 语句的功能是_____。
- (6) 下面程序的输出结果是_____。

```

#include <stdio.h>
void main( )
{
    int a,sum;
    for(a=1,sum=0;a<=100;a++)
        sum=sum+a;
}

```

```
printf("%d",sum);
}
(7) 下面程序的输出结果是_____。
```

```
void main()
{
    int a;
    for(a=1;a<5;a++)
    {
        switch(a)
        {
            case 1:printf("% d,",a);break;
            case 2:printf("% d,",a);break;
            case 3:printf("% d,",a);break;
            default:printf("OK!\n");
        }
    }
}
```

(8) 下面程序的运行结果是_____。

```
#include<stdio.h>
void main(void)
{
    int x,y;
    x=1;
    y=1;
    printf("%d,%d",x,y);
    while(x<10)
    {
        x+=y;
        y+=x;
        printf(",%d,%d",x,y);
    }
}
```

6.2 判断题(判断下列叙述的正确性,正确的请打“√”,错误的请打“×”)

- (1) while 循环是先判断表达式,后执行循环体。 ()
- (2) do-while 和 for 循环均是先执行循环体,后判断表达式。 ()
- (3) for 循环只能用于循环次数已经确定的循环。 ()
- (4) 语句 while(x!=0)…等价于语句 while(x)…。 ()
- (5) break 语句不能用于循环语句和 switch 语句之外的任何其他语句中。 ()

6.3 选择题

- (1) 下面有关 for 循环的正确描述是()。



- A. for 循环只能用于循环次数已经确定的情况
 B. for 循环是先执行循环体语句,后判断表达式
 C. 在 for 循环中不能用 break 语句跳出循环体
 D. for 循环的循环体语句中,可以包含多条语句,但必须用大括号括起来
- (2) 下面程序的运行结果是()。

```
#include <stdio.h>
void main( )
{
    int i=0,j=9,k=3,s=0;
    for(;;)
    {
        i+=k;
        if(i>j)
            break;
        s+=i;
    }
    printf("%d",s);
}
```

- A. 死循环,无输出 B. 30 C. 18 D. 3

(3) 与语句 while(!E);中的表达式!E 等价的表达式是()。

- A. E==0 B. E!=1 C. E!=0 D. E==1

(4) 设有程序段

```
int k=10;
while(k==0)
    k=k-1;
```

则下面描述中正确的是()。

- A. while 循环执行 10 次 B. 循环是死循环
 C. 循环体语句一次也不执行 D. 循环体语句执行一次

(5) 下面程序段的运行结果是()。

```
a=1;b=2;c=2;
while(a<b<c)
{
    t=a;
    a=b;
    b=t;
    c--;
}
printf("%d,%d,%d\n",a,b,c);
```

- A. 1,2,0 B. 2,1,0 C. 1,2,1

D. 2,1,1



单元7 项目的整体框架设计

能力目标

- 掌握无参函数和有参函数的定义和调用方法,能正确定义形参与实参。
- 能正确设计函数的类型和返回值类型,正确运用 return 语句实现数据返回操作。
- 能正确使用函数原型声明函数。
- 掌握函数的嵌套调用和递归调用方法,能用“传值”方式设计函数。
- 能用多种方法进行项目的整体框架设计。

知识目标

- 理解模块化程序设计的思想和函数在模块化程序设计中的地位。
- 理解不同类型函数定义的不同格式,理解函数声明的必要性。
- 理解函数类型与返回值的关系,理解形参与实参的概念,理解“传值”的特点。
- 理解局部(内部)变量与全局(外部)变量的概念与定义特点。

学习提示

函数是学习 C 语言程序设计的重要内容之一,也是学习的难点之一。

函数是模块化程序设计的最小单位,一般一个函数只完成一项功能。学习时要注意函数的类型、定义、调用、原型、形参与实参相互传递等内容,注意函数中使用的局部变量和全局变量的区别。

7.1 任务9:项目的整体框架设计

项目的整体框架设计是程序开发中关系重大的一环。整体框架是程序的总体结构,是程序设计中非常重要的部分。整体框架设计的好处是为项目搭好一个骨架,这个骨架包含了项目的各种功能模块,后面的工作就是如何完成这些功能模块,当这些功能模块全部实现后,整个项目也就完成了。

该任务是对任务8作进一步完善,实现的目标是:①主函数的位置可以任意放置;

②将原来显示一句话的语句和 getch() 函数调用语句放置到相应函数中;③增加光标定位函数;④完善程序说明函数。程序运行结果与任务 8 相似。

```
#include <stdio.h>
#include <conio.h>
#include <windows.h>
#include <stdlib.h>

void StuCover();           //项目封面函数声明
void MainMenu();          //主菜单函数声明
void EditMenu();          //编辑子菜单函数声明
void DispMenu();          //显示子菜单函数声明
void CompMenu();          //计算子菜单函数声明
void SortMenu();          //排序子菜单函数声明
void Open();              //打开文件函数声明
void Save();              //保存文件函数声明
void Add();               //增加学生记录函数声明
void Del();               //删除学生记录函数声明
void Modify();            //修改学生记录函数声明
void DispOne();           //查看一个记录函数声明
void DispAll();           //显示全部记录函数声明
void AsceSort();          //按升序排列函数声明
void DropSort();          //按降序排列函数声明
void NotElig();           //查找不及格记录函数声明
void CompSum();           //计算总成绩和平均成绩函数声明
void SearchMax();         //查找最高成绩函数声明
void SearchMin();         //查找最低成绩函数声明
void Explain();           //程序说明函数声明
void Quit(int);           //退出函数声明
void gotoxy(int x,int y); //光标定位函数声明

void main()
{
    int choose,editnum,dispnum,compnum,sortnum;//定义 5 个输入变量
    system("cls");
    StuCover();
    getch();
    while(1)                //外循环
    {
        MainMenu();         //调用主菜单函数
```



```
printf("\t\t请输入序号:");
scanf("%d",&choose);
switch(choose) //主菜单的 switch
{
case 1:Open();break;
case 2:Save();break;
case 3:
do//内循环 1
{
EditMenu();//调用编辑子菜单函数
printf("\t\t请输入序号:");
scanf("%d",&editnum);
switch(editnum)//编辑子菜单 switch
{
case 1:Add();break;
case 2:Del();break;
case 3:Modify();break;
case 0:Quit(0);break;
} //编辑子菜单 switch 结束
}while(editnum!=0); //内循环 1 结束
break;
case 4:
do//内循环 2
{
DispMenu(); //调用查看子菜单函数
printf("\t\t请输入序号:");
scanf("%d",&dispnum);
switch(dispnum) //显示子菜单 switch
{
case 1:DispOne();break;
case 2:DispAll();break;
case 3:
do//内循环 3
{
SortMenu(); //调用排序子菜单函数
printf("\t\t请输入序号:");
scanf("%d",&sortnum);
switch(sortnum) //排序子菜单 switch
{
```

```

        case 1:AsceSort();break;
        case 2:DropSort();break;
        case 0:Quit(0);break;
    }//排序子菜单 switch 结束
}while(sortnum!=0);//内循环 3 结束
break;
case 4:NotElig();break;
case 0:Quit(0);break;
} //显示子菜单 switch 结束
}while(disppnum!=0); //内循环 2 结束
break;
case 5:
do//内循环 4
{
    CompMenu(); //调用计算子菜单函数
    printf("\t\t 请输入序号:");
    scanf("%d",&compnum);
    switch(compnum) //计算子菜单 switch
    {
        case 1:CompSum();break;
        case 2:SearchMax();break;
        case 3:SearchMin();break;
        case 0:Quit(0);break;
    } //计算子菜单 switch 结束
}while(compnum!=0); //内循环 4 结束
break;
case 6:Explain();break; //程序说明
case 0:Quit(1);break;
} //主菜单的 switch 结束
} //外循环结束

void Open()//打开文件函数
{
    printf("打开文件!\n");getch();
}
void Save()//保存文件函数
{
    printf("保存文件!\n");getch();
}

```



```
}  
void Add()//增加学生记录函数  
{  
    printf("增加记录!\n");getch();  
}  
void Del()//删除学生记录函数  
{  
    printf("删除记录!\n");getch();  
}  
void Modify()//修改学生记录函数  
{  
    printf("修改记录!\n");getch();  
}  
void DispOne()//查看一个记录函数  
{  
    printf("查看选定记录!\n");getch();  
}  
void DispAll()//显示全部记录函数  
{  
    printf("显示全部记录!\n");getch();  
}  
void AsceSort()//按升序排列函数  
{  
    printf("按升序排序!\n");getch();  
}  
void DropSort()//按降序排列函数  
{  
    printf("按降序排序!\n");getch();  
}  
void NotElig()//显示不及格记录函数  
{  
    printf("显示不及格记录!\n");getch();  
}  
void CompSum()//计算总成绩和平均成绩函数  
{  
    printf("计算总成绩和平均成绩!\n");getch();  
}  
void SearchMax()//查找最高成绩函数  
{
```



```
        printf("计算最高分!\n");getch();
    }
void SearchMin()//查找最低成绩函数
{
    printf("计算最低分!\n");getch();
}

void Explain()//程序说明
{
    system("cls");
    gotoxy(10,3);//光标定位函数
    printf("这是一个教学程序。它以开发班级学生成绩管理系统为主要项目,");
    gotoxy(10,5);
    printf("旨在通过简单学生成绩管理系统软件的开发,使读者了解并掌握用C语");
    gotoxy(10,7);
    printf("言开发程序的方法与技巧。");
    gotoxy(10,9);
    printf("    该项目由 15 个任务来完成,将 C 语言基本知识与理论融入到任务中,");
    gotoxy(10,11);
    printf("完成 15 个任务后就完成整个项目的设计。通过任务驱动和项目导向教学,");
    gotoxy(10,13);
    printf("最终实现教学目的,达到培养目标。");
    gotoxy(10,15);
    printf("    该项目实施贯穿在整个教学过程中,它将重点与难点分散在各个任务");
    gotoxy(10,17);
    printf("中,达到循序渐进、逐个突破的目的,教学最后将安排一定的时间归纳汇");
    gotoxy(10,19);
    printf("总。");
    getch();
}

void gotoxy(int x,int y)
{
```



```
COORD c;
c.X=x-1;
c.Y=y-1;
SetConsoleCursorPosition(GetStdHandle(STD_OUTPUT_HANDLE),c);
}
```

前面已经出现过的函数这里不再给出。

单元能力训练任务分别如下。

任务 A: 正确定义、调用和声明有参和无参函数。

任务 B: 正确定义函数的类型, 掌握函数类型与返回值类型的关系。

任务 C: 正确设计程序说明函数和对“班级学生成绩管理系统”进行整体框架设计。

任务 D: 模仿任务 9 对“学生通讯录”的整体框架进行设计。

7.2 必备知识与理论

7.2.1 结构化程序设计思想与函数分类

1. 结构化程序设计思想

从软件开发的角度来看, 程序设计大致经历了四个阶段: 无序态程序设计、过程化程序设计、结构化程序设计和面向对象的程序设计。

结构化程序设计是以模块化设计为中心, 是将待开发的软件系统划分为若干个相互独立的模块, 由这些独立的模块完成不同的功能, 然后将这些模块通过一定的方法组织起来, 成为一个整体, 这就是结构化程序设计的思想, 其中模块化是结构化程序设计的核心。

这种设计思想很像搭积木, 单个的积木就像是一个个模块, 它们的功能单一、使用方便, 多个不同的积木按照不同的组合可以形成不同的图案。

C 语言是由函数来实现模块化设计的。函数是模块化程序设计的最小单位, 它是程序功能的载体, 函数在一般情况下要求完成的功能单一, 这样做的好处是便于函数设计与重用, 一般由主函数来完成模块的整体组织。所以设计 C 语言程序, 实际上就是设计 C 语言函数。

2. 函数的分类

C 语言函数从不同的角度可以将其分为不同类型的函数。搞清楚函数的分类, 有利于使用别人设计的函数和设计出自己需要的函数。

(1) 从定义的角度分为库函数与用户自定义函数。

库函数由系统定义, 用户只能使用(调用)。如前面学过的输出函数 `printf()` 和输入函数 `scanf()` 等都是库函数。系统为我们提供了大量的库函数, 为程序设计带来极大的方便, 但是我们大量需要的还是用户自己定义的函数。对于该类函数, 要先定义, 然后才能使用, 即“先定义后使用”, 实际上前面讲的变量和字符常量都要遵循这个原则, 后面学

习的构造类型还要遵循这个原则。

(2) 按函数有无形式参数分为有参函数与无参函数。

有参函数:调用函数时,在主调函数与被调函数之间有数据传递。

无参函数:调用函数时,主调函数与被调函数之间没有数据传递。

(3) 按函数调用关系分为主调函数和被调函数。

按结构化程序设计的思想,要求结构化模块内部的内聚力很强,结构化模块之间的联系越少越好,由于模块是由一个或多个函数组成的,那么函数之间是怎样联系的呢?它们是通过函数调用来实现函数之间的联系,因此,函数之间是可以互相调用的,调用方称为主调函数,被调用方称为被调函数。主函数可以调用函数,任何函数都不能调用主函数,主函数由系统调用。

7.2.2 函数的定义与调用

1. 函数的定义

1) 定义

任何函数(包括主函数 main())都是由函数说明和函数体两部分组成。根据函数是否需要参数,可将函数分为无参函数和有参函数两种。

(1) 无参函数的定义格式。

```
函数类型  函数名([void])
{
    说明语句部分;
    可执行语句部分;
}
```

(2) 有参函数的定义格式。

```
函数类型  函数名(数据类型 参数 1[,数据类型 参数 2...])
{
    说明语句部分;
    可执行语句部分;
}
```

有参函数比无参函数多一个参数表。调用有参函数时,调用函数将赋予这些参数实际的值。为了与调用函数提供的实际参数区别开,将函数定义中的参数表称为形式参数表,简称形参表。

【例 7.1】 定义一个函数,用于求两个数中的大数。

```
1 #include <stdio.h>
2 int max(int n1, int n2) /*定义一个函数 max()*/
3 {
```

```

4     return (n1>n2?n1:n2);
5 }
6
7 void main( )
8 {
9     int max(int n1, int n2); /* 函数说明*/
10    int num1,num2;
11    printf("input two numbers: ");
12    scanf("%d%d", &num1, &num2);
13    printf("max=%d\n", max(num1,num2));
14 }

```

程序运行结果:

```

input two numbers:45 12<回车>
max=45

```

第8、9行为说明语句部分,用来说明(定义、声明)函数和变量。第10~13行为执行语句部分。

2) 说明

函数不允许嵌套定义。

在C语言中,所有函数(包括主函数main())都是平行的。一个函数的定义,可以放在程序中的任意位置,即放在主函数main()之前或之后都可以。但在一个函数的函数体内,不能再定义另一个函数,即不能嵌套定义。

111

2. 函数的返回值与函数类型

C语言的函数兼有其他语言中的函数和过程两种功能,从这个角度看,又可把函数分为有返回值函数和无返回值函数两种。

1) 函数返回值与 return 语句

函数的返回值,是通过函数中的 return 语句来获得的。

(1) return 语句格式:

```
return (返回值表达式);
```

(2) return 语句的功能:函数执行结束,返回调用函数,并将“返回值表达式”的值带给调用函数。

注意:被调用函数中无 return 语句,并不是不返回一个值,而是返回一个不确定的值。为了明确表示不返回值,可以将函数类型定义为“void”,表示为“无(空)类型”。

2) 函数类型

在定义函数时,对函数类型的说明,应与 return 语句中返回值表达式的类型一致,也

就是说函数的类型是函数返回值的类型,它可以是我们已经学习过的 `int`、`char`、`float`、`double` 中的任意类型,也可以是要我们在后面学习的构造数据类型和指针类型。如果不一致,则以函数类型为准;如果缺省函数类型,则系统默认的函数类型为整型。

【例 7.2】 求参数 `n` 的平方。

```
double sqare(double n)
{
    return n*n;
}
```

【例 7.3】 判断参数 `n` 是否为正数。

```
int isPositive(int n)
{
    if(n>0)
        return 1;
    else
        return 0;
}
```

注意函数 `isPositive`,虽然没有错,但显得很笨拙,更好的方法是:

```
int isPositive(int n){return n>0;}或
int isPositive(int n){return(n>0)?1:0;}
```

【例 7.4】 再举一个不带参数没有返回值的例子。

```
void hello()
{
    printf("Hi!\n");
}
```

这个函数完成输出 `Hi` 并执行回车换行操作。

良好的程序设计习惯:为了使程序具有良好的可读性并减少出错,凡不要求返回值的函数都应定义为空类型,即使函数类型为整型,也不使用系统的缺省处理。

想一想,上述三个实例,能直接运行出结果吗?如果不能直接运行,该如何处理?

3. 函数原型与函数声明

1) 函数原型

编译系统在处理调用时,必须从程序中获得完成函数调用所必需的接口信息,用于确认函数调用在语法及语义上的正确性,用于判断是否有传递的参数或返回值的的数据类型,从而生成正确的函数调用代码。函数接口信息包括:函数名、函数类型、函数有无参数、参数的类型、个数与顺序等。提供接口信息任务一般由函数原型来担任。函数原型的基本格式为:

函数类型	函数名(数据类型	参数名 1[,数据类型	参数名 2,⋯]);
------	----------	-------------	------------

函数原型的格式就是在函数定义格式的基础上去掉了函数体,再加上分号构成的。也可以去掉参数表中的参数名,即:

```
函数类型 函数名 (数据类型[,数据类型,...]);
```

2) 函数声明

函数调用的接口信息必须在调用函数前提供,因此,函数原型必须位于对该函数的第一次调用处之前,一般将函数原型放在程序的开始位置。

C语言同时又规定,在以下两种情况下,可以省去对被调用函数的说明。

(1) 当被调用函数的函数定义出现在调用函数之前时。因为在调用之前,编译系统已经知道了被调用函数的函数类型、参数个数、类型和顺序。可见函数定义也兼有提供接口信息的功能。

(2) 如果在所有函数定义之前,在函数外部(例如文件开始处)预先对各个函数进行了声明,则在调用函数中可缺省对被调用函数的声明。

4. 函数的调用

在程序中,是通过函数的调用来执行函数体的,其过程与其他语言的子程序调用相似。

C语言中,函数调用的一般格式为:

```
函数名([实际参数表]);
```

切记:实际参数的个数、类型和顺序,应该与被调用函数所要求的参数个数、类型和顺序一致,才能正确地进行数据传递。

1) 函数调用方式

在C语言中,可以用以下几种方式调用函数。

(1) 函数表达式。函数作为表达式的一项,出现在表达式中,以函数返回值参与表达式的运算。这种方式要求函数有返回值。例如:

```
result=sqare(5.0);
result=sqare(5.0)+sqare(6.0);
```

(2) 函数语句。C语言中的函数可以只进行某些操作而不返回函数值,这时的函数调用可作为一条独立的语句。在这种情况下,被调用函数可以没有返回值,如果有也舍弃不用。作为函数语句调用的功能是通过函数的副作用体现的,因此把一个没有副作用的函数作为语句来调用是毫无意义的。

```
hello(); /*函数的副作用,函数输出 Hi*/
sqare(5.0); /*返回值没起作用*/
```

(3) 函数作实参。函数作为另一个函数调用的实际参数出现。这种情况是把该函数的返回值作为实参进行传递,因此要求该函数必须是有返回值的。函数作实参调用实质上是一种表达式调用。例如:

```
printf("%d\n",sqare(5.0)); /*输出 5.0 的平方值*/
```

```
printf("max=%d\n", max(num1,num2));/*输出 num1 和 num2 中大的那一个*/
```

2) 说明

- (1) 调用函数时,函数名称必须与具有该功能的自定义函数名称完全一致。
- (2) 实参在类型上按顺序与形参必须一一对应和匹配。如果类型不匹配,C 语言编译程序将按赋值兼容的规则进行转换。如果实参和形参的类型不相互兼容,通常并不给出错信息,且程序仍然继续执行,只是得不到正确的结果。
- (3) 如果实参表中包括多个参数,对实参的求值顺序随系统而异。有的系统按自左向右顺序求实参的值,有的系统则相反。Turbo C 和 VC 是按自右向左的顺序进行的。

7.2.3 函数的嵌套调用和递归调用

1. 函数的嵌套调用

函数的嵌套调用是指,在执行被调用函数时,被调用函数又调用了其他函数。如图 7.1 所示。

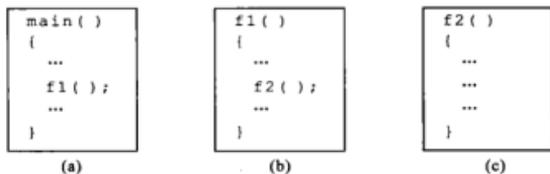


图 7.1 函数的嵌套调用格式

【例 7.5】 计算 5 的 3 次方。

```

1 double sqare(double);
2 double cubic(double n)
3 {
4     return n*sqare(n);
5 }
6
7 double sqare(double n)
8 {
9     return n*n;
10 }
11
12 #include <stdio.h>
13 int main()
14 {
15     printf("%d 的 3 次方=%.2f\n",5,cubic(5));

```

```
16     return 0;
```

```
17 }
```

程序运行结果:

5 的 3 次方=125.00

其执行过程如图 7.2 所示。



图 7.2 函数的嵌套调用执行过程

【例 7.6】 计算 $s=1^k+2^k+3^k+\dots+N^k$ 。

```

1 #define K 4
2 #define N 5
3 long f1(int n,int k) /*计算 n 的 k 次方*/
4 {
5     long power=n;
6     int i;
7     for(i=1;i<k;i++)
8         power*=n;
9     return power;
10 }
11
12 long f2(int n,int k) /*计算 1 到 n 的 k 次方之累加和*/
13 {
14     long sum=0;
15     int i;
16     for(i=1;i<=n;i++)
17         sum+=f1(i, k);
18     return sum;
19 }
20
21 #include <stdio.h>
22 void main( )
23 {
24     printf("Sum of %d powers of integers from 1 to %d=",K,N);
25     printf("%d\n",f2(N,K));
  
```

26 }

程序运行结果:

Sum of 4 powers of integers from 1 to 5=979

2. 函数的递归调用

函数的递归调用是指,一个函数在它的函数体内,直接或间接地调用它自身。

C 语言允许函数的递归调用。在递归调用中,调用函数又是被调用函数,执行递归,函数将反复调用其自身,每调用一次就进入新的一层。

为了防止递归调用无终止地进行,必须在函数内有终止递归调用的手段。常用的办法是加条件判断,满足某种条件后就不再作递归调用,然后逐层返回。

【例 7.7】 设计函数 power(n),它计算并返回 n 的阶乘,用递归方法实现。

分析:一个正整数 n 的阶乘可表示为 n!,并特别规定 0!=1。

例如,5!=1*2*3*4*5=120。

阶乘的定义还可以表示为:

$$n! = \begin{cases} 1 & n=0 \text{ 或 } n=1 \\ n * (n-1)! & n>1 \end{cases}$$

上面的算法是阶乘的递归算法,因为在求解阶乘 n! 时必须先求解另一个阶乘 (n-1)!。满足递归算法的三个条件如下。

(1) 有明确的结束递归的条件。如当 n=0 或 n=1 时,在该条件下可以直接得出 n! 等于 1。

(2) 要解决的问题总是可以转化为相对简单的同类型的问题。n! 可转化为: n * (n-1)!, 而 (n-1)! 是比 n! 稍简单的同类型问题。

(3) 随着问题的逐次转换,最终能达到结束递归的条件。算法中的参数 n 在递归过程中逐次减少,必然会达到 n=0 或 n=1。

下面 power 函数就是用递归算法实现的。

```
1 #include <stdio.h>
2 void main()
3 {
4     long power(int n);
5     int n;
6     printf("input a integer number: ");
7     scanf("%d",&n);
8     printf("%d!=%Ld\n",n, power(n));
9 }
10
11 long power(int n)
12 {
13     long f;
14     if(n>1)
```



```

15         f=power(n-1L)* n;
16     else f=1L;
17     return(f);
18 }

```

程序运行结果:

```

input a integer number:4<回车>
4!=24

```

当输入 4 时,主函数调用 $\text{power}(4)$,求 $4!$ 的示意如图 7.3 所示。右侧一系列向下的箭头表示函数调用,旁边的数字表示传递的参数;左侧向上的箭头表示返回,旁边的数字表示返回值。 power 反复调用自身: $\text{power}(4)$ 调用 $\text{power}(3)$, $\text{power}(3)$ 调用 $\text{power}(2)$, $\text{power}(2)$ 调用 $\text{power}(1)$ 。参数逐次变小,当最后调用 $\text{power}(1)$ 时,结束递归的条件已经满足,于是开始逐级完成乘法运算,最后计算出 $4!$ 的结果为 24。

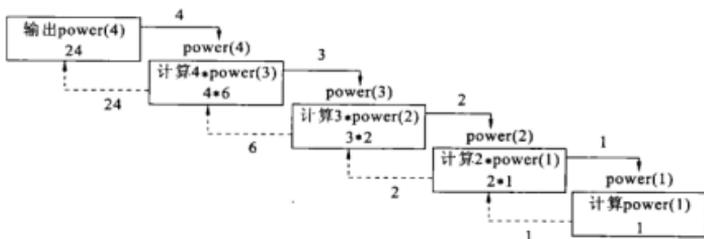


图 7.3 函数的递归调用示意图

当然在调用的过程中是需要内存开销的。

7.2.4 函数调用中的参数传递

1. 函数的形参与实参

函数的参数分为形参和实参两种,作用是实现数据在函数之间传送。

形参出现在函数定义中,只能在该函数体内使用,实参出现在调用函数中,实参应有值。发生函数调用时,调用函数把实参的值复制 1 份,传递给被调用函数的形参,从而实现调用函数向被调用函数的数据传送。

【例 7.8】 实参对形参的数据传递(无返回值)。

```

1 #include <stdio.h>
2 void main()
3 {
4     void s(int n);          /*说明函数*/
5     int n=100;             /*定义实参 n,并初始化*/
6     s(n);                  /*调用函数*/
7     printf("n_s=%d\n",n); /*输出调用后实参的值,便于进行比较*/

```

```

8 }
9
10 void s(int m)
11 {
12     int i;
13     printf("m=%d\n",m);    /*输出改变前形参的值*/
14     for(i=m-1;i>=1;i--)
15         m=m+i;            /*改变形参的值*/
16     printf("m=%d\n",m);    /*输出改变后形参的值*/
17 }

```

程序运行结果:

```

m=100
m=5050
n_s=100

```

【例 7.9】 实参对形参的数据传递(有返回值)。

```

1 #include <stdio.h>
2 void main()
3 {
4     int max(int,int);
5     int a,b,c;
6     scanf("%d,%d",&a,&b);
7     c=max(a,b);
8     printf("Max is %d\n",c);
9 }
10
11 int max(int x,int y)
12 {
13     int z;
14     z=x>y?x:y;
15     return z;
16 }

```

程序运行结果:

```

5,7<回车>
Max is 7

```

说明:

(1) 实参可以是常量、变量、表达式、函数等。无论实参是何种类型的量,在进行函数调用时,它们都必须具有确定的值,以便把这些值传送给形参。因此,应预先用赋值、输入等办法,使实参获得确定的值。

(2) 形参变量只有在被调用时,才分配内存单元,调用结束后,即刻释放所分配的内存。

存单元。因此,形参只有在该函数内有效。调用结束后,则不能再使用该形参变量。

(3) 实参对形参的数据传送是单向的,即只能把实参的值传送给形参,而不能把形参的值反向地传送给实参。如图 7.4 所示。

(4) 实参和形参占用各自不同的内存单元,即使同名也互不影响。

(5) 实参与形参的类型应当相同或赋值兼容。

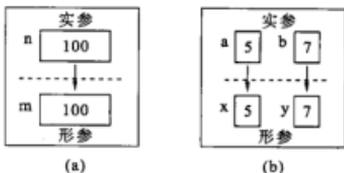


图 7.4 实参向形参传递数据示意图

2. “传值”是 C 语言传递数据的基本方式

C 语言中的函数以“传值”的方式传递参数,即只把实参表达式的值传递给被调用函数中的对应形参变量。例如,当实参是一个变量时,传递给函数的只是该变量的值,换句话说,对应形参变量所获得的值只是实参变量值的一个副本。在这种情况下,即便在函数中有改变形参变量的值的操作,也只改变实参变量的副本,而不会(也不可能)使实参变量本身的值有所变动。例 7.8 就充分说明了这一点。

【例 7.10】 计算一个数的 2 倍,考查形参结果能否影响实参?

```

1 float twice(float n)
2 {
3     return n*=2;
4 }
5
6 #include <stdio.h>
7 void main()
8 {
9     float m=7.0;
10    printf("%f\n",m);
11    printf("%f\n",twice(m));
12    printf("%f\n",m);
13 }
```

想一想:输出结果是什么? 改变了实参的值吗?

实参除了可以进行“值传递”之外,还能进行地址值的传递。关于地址值的传递将放在后面介绍。

7.3 扩展知识与理论

7.3.1 变量的作用域

C 语言中所有的变量都有自己的作用域。什么是变量的作用域? 打一个比方:每一

个变量好比是一盏灯,它所能照亮的区域就是它的“作用域”,在该区域内的任何地方都能“看到”它,自然也就能访问到该变量,出了这个区域就“看不到了”,因此也就访问不到了。

程序中每个“变量灯”的“功率”大小不一样,它们能照亮的范围也不一样,因此作用域就不一样。由什么来决定作用域的大小呢? C语言规定:变量定义的位置不同,其作用域就不同,据此将C语言中的变量分为局部变量和全局变量两大类。

1. 局部变量

定义在函数内部的变量称为局部变量(也可以称为内部变量),其作用域为块作用域,即只允许定义该变量的块中的语句访问该变量。更准确地说,块作用域的范围是从变量定义处开始,到块的结束处为止。这里所说的块主要是指复合语句。因此,定义于一复合语句中的变量,不但其他函数不能访问,该复合语句以外的语句也不能访问,即便是同一复合语句中的语句,如果它位于变量定义处之前,也不能访问该变量(一样遵循先定义后使用的原则)。这就是“局部”的含义。

例如:

```
void main ( )
{
    int x=5;
    {
        int y=10;
        printf("y=%d\n",y); } y的作用域
    }
    printf("x=%d\n",x);
    printf("y=%d\n",y); /*编译时将报错*/
}
```

} x的作用域

函数的函数体就是一个复合语句,因此一种典型的情况就是在函数体的开始处(同时也是复合语句开始处)定义的变量,该函数中的任何语句都可以访问。

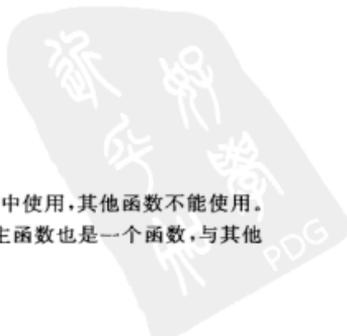
又如:

```
int f1(int a) /*函数 f1*/
{
    int b,c;
    ...
}
/*a,b,c作用域:仅限于函数 f1()中*/
```

说明:

(1) 主函数 main()中定义的内部变量,也只能在主函数中使用,其他函数不能使用。同样,主函数也不能使用其他函数中定义的内部变量。因为主函数也是一个函数,与其他函数是平行关系,应予以注意。

(2) 形参变量是局部变量,作用域被限制在函数内。



(3) 允许在不同的函数中使用相同的变量名,它们代表不同的对象,分配不同的单元,互不干扰,也不会发生混淆。

2. 全局变量

在函数外部定义的变量称为全局变量(也可以称为外部变量)。用于记录应用系统的全局信息,也是函数之间交换数据信息的媒介。

全局变量不属于任何一个函数,其作用域是:从全局变量的定义位置开始,到本文件结束为止。

全局变量作用域如下所示。

```
float u=1.5, v=3.2;
char s1(int a)
{
    float b,c;
    ...
}
int p,q;
float s2(char x,int y)
{
    int m,n;
    ...
}
main()
{
    int s,r;
    ...
}
```

全局变量 u、v 的作用域

全局变量 p、q 的作用域

121

【例 7.11】 输入长方体的长(l)、宽(w)、高(h),求长方体体积及正、侧、顶三个面的面积。

```
1 int s1,s2,s3; //全局变量
2 int vs(int a,int b,int c)
3 {
4     int v;
5     v=a*b*c; /*计算体积*/
6     s1=a*b; /*计算顶面面积*/
7     s2=b*c; /*计算侧面面积*/
8     s3=a*c; /*计算正面面积*/
9     return v;
10 }
11
```

设计知识 PDG

```

12 #include <stdio.h>
13 void main( )
14 {
15     int v,l,w,h;
16     printf("\ninput length,width and height: ");
17     scanf("%d%d%d",&l,&w,&h); //注意输入数据的格式
18     v=vs(l,w,h);
19     printf("v=%d\ns1=%d\ns2=%d\ns3=%d\n",v,s1,s2,s3);
20 }

```

程序运行结果:

```

input length,width and height:4 5 6<回车>
v=120
s1=20
s2=30
s3=24

```

说明:

(1) 全局变量可加强函数模块之间的数据联系,但又使这些函数依赖这些全局变量,因而这些函数的独立性降低。从模块化程序设计的观点来看这是不利的,因此不是非用不可时,不要使用全局变量。

(2) 在同一源文件中,允许全局变量和局部变量同名。在局部变量的作用域内,全局变量将被屏蔽而不起作用。当然系统不会混淆,并不意味着人也不会混淆,所以应该尽量少用同名变量。

(3) 全局变量的作用域是从定义点到本文件结束。如果定义点之前的函数需要引用这些全局变量时,需要在函数内对被引用的全局变量进行说明。全局变量说明的一般格式为:

```
extern 数据类型 全局变量名 1 [,全局变量名 2...];
```

全局变量的定义和全局变量的说明是两回事。全局变量的定义必须在所有的函数之外,且只能定义一次。而全局变量的说明,出现在要使用该全局变量的函数内,而且可以出现多次。

关键字“extern”用来扩展全局变量的作用域,使得以前不能访问它的函数也能访问到它,这种作用域的扩展,也称为作用域的“提升”。

【例 7.12】 全局变量的定义与说明。

```

1 int vs(int xl,int xw)
2 {
3     extern int xh; //全局变量 xh 的说明*/
4     int v;
5     v=xl*xw*xh; //直接使用全局变量 xh 的值*/
6     return v;

```

```

7 }
8
9 #include "stdio.h"
10 void main()
11 {
12     extern int xw,xh;      /*全局变量的说明*/
13     int x1=5;             /*局部变量的定义*/
14     printf("x1=%d,xw=%d,xh=%d\nv=%d\n",x1,xw,xh,vs(x1,xw));
15 }
16     int x1=3,xw=4,xh=5;   /*全局变量 x1,xw,xh 的定义*/

```

程序运行结果：

```

x1=5,xw=4,xh=5
v=100

```

请读者分析其中的原因。

7.3.2 变量的生存期

从变量的作用域的角度,可将变量分为局部变量和全局变量两种。局部变量定义在函数体内,函数被调用时,局部变量才临时地被创建,函数执行完后,局部变量自动被销毁;全局变量是定义在整个程序空间内的,在开始运行程序时被创建,整个程序执行完了才会被销毁,因此一个变量(无论是全局还是局部变量)都有一个“创建”、“生存”、“销毁”的过程,从这个意义上说,变量是有“寿命”或存在“生存期”的。

变量的生存期取决于它的存储类型。所谓“存储类型”是指系统为变量分配的具有某种特性的存储区域,存储区域一般分为两种:静态存储区和动态存储区。存放在静态存储区中的变量在程序运行初期就被创建,它们的寿命往往与程序同步;存放在动态存储区中的变量是临时性的,在程序运行期间随时会被撤销。

系统将内存中供用户使用的存储空间分为三部分,如图 7.5 所示。

程序运行时,局部变量放在动态存储区中,全局变量放在静态存储区中。

C 语言的变量(包括函数)有两个属性:数据类型和数据存储类型。数据类型大家已熟悉,数据存储类型具体包含四种,这里只介绍常用的两种。根据存储类型可以知道变量的生存期。

1. 自动变量(auto)

函数中的局部变量,如果不专门声明为 static 存储类型,都是动态地分配存储空间的,数据存储于动态存储区中。前面的例子中定义的全部局部变量实际上都是自动变量,只是省略了 auto。

自动变量定义格式为:

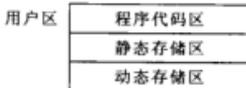


图 7.5 存储空间示意图

[auto] 类型标识符	变量名表;
--------------	-------

自动变量的存储特点如下。

(1) 自动变量属于动态存储方式,变量存放在动态存储区。在函数中定义的自动变量,只在该函数内有效,函数被调用时分配存储空间,调用结束就释放。在复合语句中定义的自动变量,只在该复合语句中有效,退出复合语句后,也不能再使用,否则将引起错误。

(2) 定义而不初始化,则其值是不确定的(随机值)。如果初始化,则赋初值操作是在调用时进行的,且每次调用都要重新赋一次初值。

(3) 由于自动变量的作用域和生存期都局限于定义它的个体内(函数或复合语句),因此不同的个体中允许使用同名的变量而不会混淆。即使在函数内定义的自动变量,也可与该函数内部的复合语句中定义的自动变量同名。

【例 7.13】 自动局部变量的定义与应用。

```

1 #include <stdio.h>
2 int SUM(int a,int b)      /* 隐式定义形参自动变量 a,b */
3 {
4     auto int c;          /* 显示的定义自动变量 c */
5     c=a+b;
6     return c;
7 }
8
9 void main( )
10 {
11     int a,b,sum;        /* 隐式的定义自动变量 a,b,sum */
12     printf("请输入两个数(中间用空格分开):");
13     scanf("%d%d",&a,&b);
14     sum=SUM(a,b);
15     printf("sum=%d\n",sum);
16 }

```

程序运行结果:

```

请输入两个数(中间用空格分开):34 56<回车>
sum=90

```

从这例子可以看出,自动局部变量的作用域与生存期都在函数内有效,离开了定义它的函数作用域和生存期就失效了。

2. 静态变量(static)

关键字“static”译成中文就是“静态的”。如果在自动局部变量前面加上一个关键字“static”,它就变成了“静态局部变量”。

静态变量定义格式为:

static 类型标识符 变量名表;

静态变量的存储特点如下。

(1) 静态局部变量属于静态存储,变量存放在静态存储区。在程序执行过程中,即使所在函数调用结束也不释放。换句话说,在程序执行期间,静态局部变量始终存在,但其他函数是不能引用它们的。

(2) 定义如果不初始化,则自动赋以"0"(整型或实型)或'\0'(字符型),且每次调用它们所在的函数时,不再重新赋初值,只是保留上次调用结束时的值。

(3) 自动局部变量与静态局部变量二者的寿命不同,自动局部变量从函数调用时开始存在,函数调用结束时自动销毁;静态局部变量从函数调用时开始存在,调用结束时不销毁,程序结束时才销毁,寿命是全局的。

需要保留函数上一次调用结束时的值时,往往使用静态局部变量。

静态局部变量与自动局部变量有什么异同呢?

相同点:都是局部变量,即作用域相同。

不同点:存储的地点不同,自动局部变量存储于动态存储区中,而静态局部变量存储于静态存储区中。

【例 7.14】 自动变量与静态局部变量的存储特性。

```

1 #include <stdio.h>
2 void auto_static(void)
3 {
4     int var_auto=0;           /* 自动变量:每次调用都重新初始化*/
5     static int var_static=0; /* 静态局部变量:只初始化 1 次*/
6     printf("var_auto=%d,var_static=%d\n",var_auto,var_static);
7     ++var_auto;
8     ++var_static;
9 }
10
11 void main()
12 {
13     int i;
14     for(i=0; i<5; i++)
15         auto_static();
16 }

```

程序运行结果:

```

var_auto=0,var_static=0
var_auto=0,var_static=1
var_auto=0,var_static=2
var_auto=0,var_static=3
var_auto=0,var_static=4

```



7.3.3 预处理命令

C语言提供的预处理命令主要有四种:不带参宏定义、带参宏定义、文件包含和条件编译。这里只介绍前三种。

预处理命令是以#号开头的代码行,每一条预处理命令必须单独占用一行,由于不是C语言的语句,因此在结尾不能有分号“;”。

1. 不带参宏定义

用一个指定的标识符(即名字)来代表一个字符串。

1) 格式

```
#define 标识符 字符串
```

如前面介绍过的符号常量的定义方法:

```
#define PI 3.1415926
```

功能:用“PI”来代替“3.1415926”这个字符串,在编译预处理时,即在编译之前将程序中在该命令以后出现的所有的“PI”都用“3.1415926”代替。这种方法使用户能以一个简单的名字来代替一个长的字符串(又称为宏体),方便了用户编程,避免了书写可能出现的错误。因此把这个标识符(名字)称为“宏名”,在预编译时将宏名替换成字符串的过程称为“宏展开”。

【例 7.15】 求圆周长、面积和球体积。

```
1 #include <stdio.h>
2 #define PI 3.14159
3 void main()
4 {
5     float r,l,s,v;           /*定义变量*/
6     printf("Input radius:");
7     scanf("%f",&r);         /*输入变量*/
8     l=2*PI*r;               /*计算圆周长*/
9     s=PI*r*r;               /*计算圆面积*/
10    v=4.0/3.0*PI*r*r*r;      /*计算圆体积*/
11    printf("L=%8.4f\nS=%8.4f\nV=%8.4f\n",l,s,v);
12 }
```

程序运行结果:

```
Input radius:5<回车>
L=31.4159
S=78.5397
V=294.5240
```



2) 说明

(1) #define, PI 和 3.14159 之间一定要有空格,而且习惯上将宏名定义成大写,以区别于变量名,但这并非规定。

(2) 宏被定义后,其作用域一般为定义它的文件,通常 #define 命令写在文件的开头,但这也并非规定,实际上宏定义可以出现在程序的任何地方,但必须位于引用之前。

(3) 宏被定义后,一般不能再重新定义,而可以用 #undef 命令提前终止宏定义的作用域,如图 7.6 所示。

(4) 一个定义过的宏名可以用来定义其他新的宏,但应当注意其中的括号。例如:

```
#define WIDTH 50
#define LENGTH (WIDTH+20)
```

宏 LENGTH 等价于:

```
#define LENGTH (50+20)
```

有没有括号意义截然不同,例如:

```
variable=LENGTH*20;
```

若宏体中有括号,则宏展开后变成:

```
variable=(50+20)*20;
```

若宏体中没有括号,则宏展开后变成:

```
variable=50+20*20;
```

显然二者的结果是不一样的。

(5) 宏定义是专门用于定义宏名的,它与定义变量的含义是不一样的,宏定义只作字符替换,不分配内存空间。

(6) 宏定义不是 C 语句,不必在行末加分号。由于是简单置换,如果加了分号则会连分号一起进行置换。如:

```
#define PI 3.1415926;
s=PI*r*r;
```

经过宏展开后,该语句为:

```
s=3.1415926;*r*r;
```

显然会出现语法错误。

由上所述可以看出,宏定义是用宏名来代替宏体的,它只做简单的替换,而不做正确性检查,使用宏定义时要特别注意这一点。另外,使用宏名来代替字符串,可以减少程序中重复书写字符串的工作量,减少书写错误的发生,同时也便于修改,真正达到一处修改处处修改的目的。

2. 带参宏定义

带参数的宏定义不仅进行简单的字符串替换,还要进行参数替换。

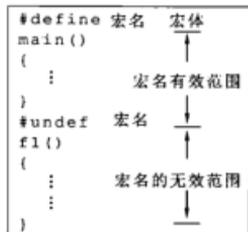


图 7.6 宏定义的作用域

1) 格式

```
#define 宏名(形参表) 宏体
```

宏名是一个标识符,形参表中可以有一个参数,也可以有多个参数,多个参数用逗号分隔。宏体是被替换的字符序列。

功能:将程序中凡出现宏名的地方均用宏体替换,并用实参代替宏体中的形参。例如:

```
#define MIN(a,b) ((a)<(b)?(a):(b))
```

其中(a,b)是宏 MIN 的参数表,如果有下面的语句:

```
min=MIN(3,9);
```

则在出现 MIN 处用宏体((a)<(b)?(a):(b))替换,并且用实参 3 和 9 去代替形参 a 和 b。

```
min=(3<9?3:9); /*结果为3*/
```

带参数的宏展开与实参替换形参如图 7.7 所示。

#define MIN(a,b)	((a)<(b)?(a):(b))
↑↑	↓
Min=MIN(3,9)	(3<9?3:9)

图 7.7 带参数宏替换示意图

很显然,带参数的宏相当于一个函数的功能,但却比函数简捷。

2) 说明

(1) 在书写带参的宏定义时,宏名与左括号之间不能出现空格,否则右边都作为宏体。例如:

```
#define MIN (a,b) ((a)<(b)?(a):(b))
```

这时将(a,b)((a)<(b)?(a):(b))作为宏名 MIN 的宏体字符串了。

(2) 由于优先级的不同,定义带参数的宏时,宏体中与参数名相同的字符序列带小括号与不带小括号意义有可能不一样。

例如:

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

```
Area=S(2,5);
```

展开后为:Area=2*5;

如果Area=S(w,w+5);

展开后 Area=w*w+5;由于乘法的优先级高于加法的优先级,显然得不到希望的值。

如果将宏定义改为:

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

无论是 Area=S(2,5);还是 Area=S(w,w+5);都将得到希望的值。

由此可以看出宏体中适当加小括号所起的作用。

(3) 带参的宏传递参数与函数调用实参与形参的传递也是有区别的,函数调用时,先

求实表达式的值,然后传递给形参。而使用带参的宏只是进行简单的字符替换,例如上面的例子就是如此。

(4) 函数的形参与实参要求类型匹配,并进行类型检查。带参的宏不存在类型问题,宏名无类型,它的参数没有类型也不进行类型检查。

【例 7.16】 带参宏定义的应用。

```
1 #include <stdio.h>
2 #define MIN(a,b) ((a)<(b)?(a):(b))
3 void main()
4 {
5     int min;
6     char ch;
7     min=MIN(5,3);
8     ch=MIN('5','3');
9     printf("min=%d\n",min);
10    printf("ch=%c\n",ch);
11    printf("ch=%s\n",MIN("5","3"));
12 }
```

程序运行结果:

```
min=3
ch=3
ch=5
```

显然,第 11 行输出语句不能得到正确的结果,这是因为字符串的比较必须用专门的比较函数,但编译时没有报错,因此编程时要特别小心。如果善于利用宏定义,可以实现程序的简化。

3. 文件包含

所谓“文件包含”是指将一个源文件的内容全部合并到当前源文件中,即将一个源文件包含到本文件中。

1) 格式

C 语言提供了 `#include` 命令用来实现文件包含的操作,它有下列两种格式:

#include <头文件名>

或

#include "头文件名"

注意:在 `include` 与 `<`或`>`之间的空格不是必须的。

在 C 语言程序设计中经常会用到系统函数,有时程序员自己也要定义宏、结构体类型、全局变量等,它们的声明往往都分门别类地放在不同的“头文件”中(以“.h”为扩展名)。前面的例子都用到 `#include<stdio.h>` 命令,这是因为标准输入 `scanf()` 和输出

printf()函数的原型都放在 stdio.h 头文件中,如果程序中要使用标准输入和输出函数,就必须用命令 #include<stdio.h>,将“stdio.h”文件的所有内容插入到当前文件中,编译时就能检查到这些函数原型的存在,否则就会报错。stdio.h 称为“标准输入输出函数头文件”。C 语言中还有很多这样的头文件,如“math.h”(常用数学运算函数头文件)、“string.h”(字符串操作函数头文件)、“stdlib.h”(一些常用的子程序头文件)、“dir.h”(目录与路径操作函数头文件)等。在第 5 章例 5.21 中也能见到用户自定义的头文件的使用例子。

文件包含的两种格式在使用时还是有一定的区别的,第一种格式中用“<>”括起来的,用来包含的那些由系统提供的头文件存放在指定目录中(一般在 include 子目录中)。第二种用双引号括起来的格式,用来包含那些由程序员自己定义的放在当前目录下的头文件。通俗地说就是如果用<>的格式,系统就在 include 目录下查找头文件。如果用"的格式,系统首先在当前目录下查找,如果找不到再到 include 目录下查找头文件。

“文件包含”命令是很有用的命令,它可以减少程序员的重复劳动。

2) 说明

(1) #include 命令只能包含一个头文件,如果想包含多个头文件,则必须用多条文件包含命令。例如:

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

(2) 使用文件包含后会使得编译后的目标文件变长,为使目标文件不至于过长,在定义包含文件时,其内容不宜过多。因为过多后,常常会使被包含文件的内容利用率下降,而很多用不到的内容增加了目标文件的长度。

【例 7.17】 本程序说明用户如何定义自己的头文件并在程序中引用。

首先创建一个用户定义的名为 usehead.h 的头文件,并存放在当前目录下。其内容为:

```
#define PRT printf
#define PI 3.14159
```

再编写一个名为 test.c 的源文件。

```
1 #include "usehead.h"           //包含用户自定义的头文件
2 #include <stdio.h>            //包含库文件
3 void main( )
4 {
5     float area;
6     int ridius;
7     PRT("ridius=");
8     scanf("%d",&ridius);
9     area=PI*ridius*ridius;
10    PRT("area=%5.2f\n",area);
11 }
```

程序运行结果:



```
radius=4<回车>
area=50.24
```

在上面的程序中,包含一个用户自定义的头文件 usehead.h,该头文件存放在当前目录下,文件中包括了两个宏定义,编译时系统会查询 PTR 和 PI 的出处,结果在 usehead.h 中找到,因此编译不会报错。如果没有在源文件中使用 #include 命令嵌入 usehead.h,则编译时一定会报错,因为系统不能确定 PRT 和 PI 的出处。

实际上多文件操作就是利用文件包含命令来实现的。

7.3.4 常见错误及处理方法

常见错误 1:函数功能设计混乱。

初学者常常会觉得设计函数是一件十分困难的工作,往往不知道怎样设计函数。这里给出一个设计函数的思考方法:一是函数设计的功能越单一越好,也就是一个函数最好只完成一件事;二是如果函数要处理数据,而这个数据又来自于其他函数,就必须设计形参,而且要根据数据的类型与个数设计形参的类型与个数;三是函数处理后的数据如果需要被其他函数使用,这个函数就应有返回值,函数的类型就应与返回值的类型相同。

常见错误 2:在“值传递”方式下改变实参的值。

由于“值传递”实参与形参的数据传递是单向传递,实参将数据传给形参,而不能将形参的值传回实参,因为实参与形参都开辟彼此独立的存储空间,函数中对形参的操作不会影响到实参,因此,不能在“值传递”方式下改变实参的值。

常见错误 3:

```
int SUM(int a,int b) //函数定义
{
    ...
}
void main()
{
    float x;//实型变量
    ...
    sum=SUM(x); //函数调用
    ...
}
```

由于函数调用要求形参与实参的类型、个数、顺序一致,上述实例中形参与实形的类型、个数、顺序均不一致,将出现“conversion from ‘float’ to ‘int’, possible loss of data”(单精度型向整型转换,可能造成数据损失)和“too few actual parameters”(实参少于两个)的提示。

常见错误 4:变量的作用域与生存期的概念模糊。

这是两个不同的概念,变量的作用域是指变量可以使用的区域,而变量的生存期是指变量在存储空间存在的时间。对于自动局部变量而言,该变量的作用域与生存期是一样的,都在函数(块)内有效,离开了函数(块),变量存储空间释放,变量不能使用;对于静态

局部变量,该变量的作用域与生存期是不一样的,静态局部变量的作用域在函数(块)内,而生存期与程序同步;对于外部变量,作用域是从定义处开始到本源文件结束,而生存期与程序同步。

常见错误 5:定义 `#define SUM(x) x * x`,调用 `SUM(10)`,结果为 100,是正确的,但如果将调用改为 `SUM(5+5)`,还希望得到 100,结果却是 35。宏展开是 `5+5 * 5+5`,计算结果是 35 而不是 100。为了避免上述宏定义的二义性错误,最好的办法是增加括号,将上述宏定义修改为 `#define SUM(x) (x) * (x)`,可以得到正确的结果。

常见错误 6:

```
#include "stdio.h"
#define a(x) (x) * (x)
main()
{
    int x=5;
    printf("%d,",a(x++));
    printf("%d\n",x);
}
```

按照编程者的本意是想得到这样的宏展开 $(x+1) * (x+1)$,得到结果为 36,6 的值,但实际执行结果却是 25,7。按宏的正确展开结果是 $(x++) * (x++)$, x 为后自增,按先使用后自增的原则, x 先进行乘法运算,即 $5 * 5$,再进行 x 两次自增 1 运算,结果自然为 25,7。如果将 `printf("%d,",a(x++))`;改为 `printf("%d,",a(++x))`;,执行结果是 49,7。因为宏展开后得到 $(++x) * (++x)$,按先自增后使用的原则, x 分别自加 1,由于 x 在内存中只有一个空间,所以两次自增操作使 x 的值变为 7,两个 7 再执行乘法运算,结果为 49,7。

7.4 深入训练

(1) 设计两个函数,一个用来计算圆的周长,另一个计算圆的面积,并在主函数中调用这两个函数。

(2) 用递归方式,求 15! 的值,要求:设计一函数计算阶乘,在主函数中输出阶乘值。

(3) 设计一程序,任意输入一个整数,求各位数之和。

(4) 设计一程序,输出 100~200 之间的所有素数。

(5) 设计一程序,判断输入的密码是否正确(正确的密码可以自由设计),若正确,显示“欢迎使用本软件!”,若不正确,可以重新输入,允许输入三次,如果三次都不正确,显示“密码不正确,你不能使用本软件”,并退出程序。

(6) 设计一程序,实现任意输入一个正整数 num ,求 $1!+3!+5!+\dots+num!$ 之和。要求将阶乘计算与求和计算分别设计成函数,主函数中输入 num 值,调用两个计算函数并输出和。

(7) 设计一程序,能根据用户要求实现不同进制(2、8、10、16)之间的数据转换。主函

数显示进制转换菜单,并调用相关函数实现转换功能。

- (8) 编写一个宏定义 MYALPHA(C)判断 C 是否是字母字符,若是则输出 1,否则输出 0。
- (9) 编写一程序,求三个数中最大者,要求用带参宏实现。
- (10) 编写一程序,输入两个整数,求它们相除的余数,并用带参的宏来实现。

习 题 7

7.1 填空题

- (1) 对于有返回值的函数,要结束函数运行必须使用_____语句。
- (2) 每一个形式参数就是一个_____。
- (3) 函数按调用关系可分为_____和_____两种。
- (4) 函数的类型就是函数_____的类型。
- (5) 函数有三种不同方式的调用,作为_____的函数调用、作为_____的函数调用和作为_____的函数调用。
- (6) 如果要自动局部变量改变成静态局部变量,则应在该变量定义前加_____修饰。
- (7) 只允许被一个特定的复合语句访问的变量称为_____变量。
- (8) 函数的递归调用指的是_____。
- (9) 在函数外部定义的变量称之为_____变量,在函数内部定义的变量称之为_____变量。
- (10) 调用函数时,实参可以是_____、_____、_____、_____。形参可以是_____和_____。
- (11) 在 C 语言中,系统将内存中供用户使用的存储空间分为三部分,它们是_____、_____、_____。
- (12) 变量的作用域是指_____,变量的生存期是指_____。
- (13) 作为_____的函数调用,即使有返回值也被舍弃不用。
- (14) 下面 add() 函数的功能是求两个参数的和,函数中错误的部分是_____,改正后为_____。
- ```
void add(float a, float b)
{
 float c;
 c=a+b;
 return (c);
}
```
- (15) 全局变量的作用域是从\_\_\_\_\_开始到\_\_\_\_\_结束。如果想提升变量的作用域可以采用\_\_\_\_\_方法。
- (16) C 语言规定,预处理命令必须以\_\_\_\_\_开头。

(17) 用来定义符号常量的预处理命令是\_\_\_\_\_。

(18) C 语言规定,一行只能出现\_\_\_\_\_预处理命令。

### 7.2 判断题(判断下列叙述的正确性,正确的请打“√”,错误的请打“×”)

(1) 简单的 C 语言源程序不需要主函数。 ( )

(2) 主函数无论在程序的什么位置,程序总是从主函数开始执行。 ( )

(3) 函数中的局部变量与全局变量同名时,局部变量优先,全局变量暂时不起作用。 ( )

(4) 有两种方法可以终止函数的运行,一种是遇到函数结束标志{}后即可终止函数运行,另一种是使用 return 语句。 ( )

(5) 全局变量可以被任何一个函数中的任何一个表达式使用。 ( )

(6) 在函数调用中,实参为表达式时,与其对应的形参也必须是表达式。 ( )

### 7.3 选择题

(1) C 语言中,决定函数返回值的类型是( )。

- A. return 语句中的表达式的类型      B. 调用函数的主调函数的类型  
C. 调用函数时临时决定的类型      D. 定义函数时所指定的函数类型

(2) (多选)每一个实参就是一个( )。

- A. 表达式      B. 常量      C. 变量      D. 函数调用

(3) 如果一个函数无返回值,则只能作为( )被调用。

- A. 表达式      B. 语句      C. 有参函数      D. 无参函数

(4) C 语言规定,调用一个函数时,实参与形参变量之间的数据传递是( )。

- A. 地址传递      B. 值传递  
C. 由实参传给形参,再由形参传给实参      D. 由用户指定传递对象

(5) 在 fun((n1,n2),(n3,n4,n5),n6); 函数调用中,含有( )个实参。

- A. 3      B. 4      C. 5      D. 6

(6) 以下错误的描述是( )。

- A. 实参可以是常量、变量、表达式      B. 形参可以是常量、变量、表达式  
C. 实参可以是任意数据类型      D. 形参应与其对应的实参类型一致

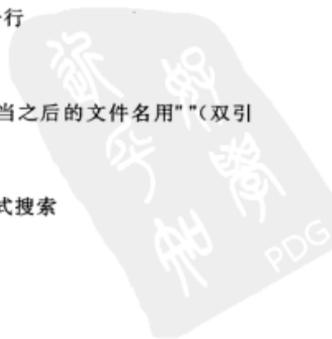
(7) 以下不正确的叙述是( )。

- A. 预处理命令行都必须以“#”号开始  
B. 在程序中凡是以“#”号开始的命令行都是预处理命令行  
C. C 语言在执行过程中对预处理命令进行处理  
D. #define ABCD 是正确的宏定义

(8) 在文件包含预处理语句(#include)的使用形式中,当之后的文件名用“(双引号)括起来,寻找包含文件的方式是( )。

- A. 直接按系统设定的标准方式搜索目录  
B. 先在源程序所在的目录搜索,再到系统设定的标准方式搜索  
C. 仅仅搜索源程序所在的目录  
D. 仅仅搜索当前目录

(9) 以下程序的输出结果是( )。



```
#include<stdio.h>
#define ADD(y) 3.54+y
#define PR(a) printf("%d", (int)(a))
main()
{
 int i=3;
 PR(ADD(3)*i);
}
```

- A. 25                      B. 20                      C. 12                      D. 10

(10) 以下程序的输出结果是( )。

```
#include<stdio.h>
#define N 2
#define M N+2
#define CUBE(x) (x*x*x)
main()
{
 int i=M;
 i=CUBE(i);
 printf("%d\n",i);
}
```

- A. 17                      B. 64                      C. 125                      D. 53

(11) 在宏定义 #define PI 3.14159 中,用宏名 PI 代替一个( )。

- A. 常量                      B. 单精度数                      C. 双精度数                      D. 字符串

(12) 用 #define 命令定义的符号常量在程序运行期间( )。

- A. 可以再次赋值,使用方法如同变量                      B. 不可改变  
C. 不能确定                      D. 以上的说法都不正确

(13) 下列程序的运行结果是( )。

```
#include "stdio.h"
#define X 5
#define Y X+1
#define Z Y*X/2
main()
{
 int a;
 a=Y;
 printf("%d,",Z);
 printf("%d\n",--a);
}
```

- A. 15,5                      B. 15,6                      C. 7,5                      D. 7,6



## 单元 8 项目中数组的应用

### 能力目标

- 能正确定义数值型一维数组,能用循环实现对一维数组元素的访问。
- 能正确使用数组名作函数参数,传递数组首地址并进行数组元素的正确访问。
- 能正确定义字符数组、能正确初始化字符数组。
- 能正确引用单个字符数组元素和整体引用字符数组。
- 能正确使用常用的字符数组处理库函数,能正确处理带空格字符串与不带空格字符串的输入。

### 知识目标

- 理解一维数组的概念与定义方法、理解一维数组元素的下标访问法。
- 理解一维数组在内存中的存储结构、理解传地址与传值的区别与联系。
- 理解字符数组与字符串的区别与联系、理解数值型数组与字符型数组的区别与联系。
- 理解数值型数据的赋值操作与字符数组的赋值操作的区别。
- 理解字符数组名作函数参数与字符串作函数参数的区别与联系。

### 学习提示

在现实社会中有大量类型相同的数据,如学生学号(整型)、成绩(实型)、姓名(字符型)等,C语言用数组来简单、快捷处理这些类型相同的若干数据。学习这个内容时我们要注意区分数值型数组与字符型数组的区别与联系,学会数组元素的访问方法(下标法)和字符型数组的整体访问方法。另外,学习字符串处理库函数时要特别注意函数的参数和返回值。

### 8.1 任务 10:初步完善学生最高、最低等成绩查找

该任务初步实现项目中查找最高学生成绩函数 SearchMax()、查找最低学生成绩函数 SearchMin()、查找不合格成绩函数 NotElig()。

要完成上述任务,在主函数中,我们设计一个包含 10 个学生成绩的实型数组。定义数组的整型常量表达式用符号常量表示,然后完善相应函数。在编写函数的过程中要十分注意函数的形参与实参的设计。

- (1) 在项目的文件包含行的下面增加下列内容:

```
#define STUSIZE 10
```

- (2) 相应函数声明修改为:

```
void SearchMax(float score[],int stusize); //查找最高学生成绩函数声明
void SearchMin(float score[],int stusize); //查找最低学生成绩函数声明
void NotElig(float score[],int stusize); //查找不合格学生成绩函数声明
```

- (3) 主函数稍做修改:

```
void main()
{
 int choose,editnum,dispnum,compnum,sortnum; //定义 5 个输入变量
 float stuscore[STUSIZE]={65.5,80,97.5,55,85,77.5,89,95,68.5,88};
 //定义学生成绩数组
 ...
}
```

- 三个函数的调用语句修改为:

```
SearchMax(stuscore,STUSIZE);
SearchMin(stuscore,STUSIZE);
NotElig(stuscore,STUSIZE);
```

- (4) 三个函数定义修改为:

```
void SearchMax(float score[],int stusize) //查找最高学生成绩函数
{
 float max=score[0]; //假定第一个数为最大
 int i,flag;
 system("cls");
 for (i=1;i<stusize;i++) //max 依次与数组中的其他数据比较
 if (max<score[i])
 {
 max=score[i]; //将比 max 大的数据送给 max
 flag=i; //记录最大值的下标
 }
 gotoxy(20,5);
 printf("成绩最高的是:%.1f\n",score[flag]); //输出下标为 flag 的元素
 gotoxy(20,10);
 printf("查找最高分成功,按任意键返回上级菜单!");
 getch();
}
```

```
void SearchMin(float score[],int stusize) //查找最低学生成绩函数
{
 float min=score[0];
 int i,flag;
 system("cls");
 for(i=1;i<stusize;i++)
 if(min>score[i])
 {
 min=score[i];
 flag=i;
 }
 gotoxy(20,5);
 printf("成绩最低的是:%.1f\n",score[flag]);
 gotoxy(20,10);
 printf("查找最低分成功,按任意键返回上级菜单!");
 getch();
}
```

```
void NotElig(float score[],int stusize) //查找不合格学生成绩函数
{
 int i,flag=0;
 system("cls");
 gotoxy(20,5);
 printf("不合格成绩: ");
 for(i=0;i<stusize;i++)
 if(score[i]<60)
 {
 printf("%6.1f",score[i]);
 flag=1;
 }
 if(!flag)
 {
 gotoxy(35,5);
 printf("没有不合格成绩!");
 }
 gotoxy(20,10);
 printf("查找不合格成绩成功,按任意键返回上级菜单!");
 getch();
}
```



从 SearchMax 函数和 SearchMin 函数的程序代码可以看出,它们除了循环中的判断条件不相同外,绝大部分是相同的。我们只要稍加修改,就可以将这两个函数改写成一个函数,这样既减轻了编程者的劳动强度,又提高了程序的阅读性。

下面将 SearchMax 函数和 SearchMin 函数优化合并成一个函数。

```
void SearchMaxMin(float score[],int stusize,int n) //查找最高和最低成绩函数
```

```
{
 float maxmin=score[0];
 int i,flag;
 system("cls");
 for(i=1;i<stusize;i++)
 {
 if((n==1)? (maxmin<score[i]) : (maxmin>score[i]))
 {
 maxmin=score[i];
 flag=i;
 }
 }
 if(n==1)
 {
 gotoxy(20,5);
 printf("成绩最高的是:%.1f\n",score[flag]);
 gotoxy(20,10);
 printf("查找最高分成功,按任意键返回上级菜单!");
 }
 else
 {
 gotoxy(20,5);
 printf("成绩最低的是:%.1f\n",score[flag]);
 gotoxy(20,10);
 printf("查找最低分成功,按任意键返回上级菜单!");
 }
 getch();
}
```

在这个函数中,我们增加了一个形参,根据形参的值来确定是求最高还是求最低学生成绩(1 为求最高成绩,0 为求最低成绩),然后,用条件表达式来判断是执行求最高成绩比较还是执行求最低成绩比较。函数调用也相应修改为:

查找最高成绩:SearchMaxMin(stuscore,STUSIZE,1);

查找最低成绩:SearchMaxMin(stuscore,STUSIZE,0);

对于函数声明部分,应当先删除原来的两个函数声明,重新增加新的函数声明:

```
void SearchMaxMin(float score[],int stusize,int n); //查找最高和
最低成绩函数
```

该函数还可以进一步优化,使之更为简捷,读者不妨一试。

单元能力训练任务分别如下。

任务 A:正确定义含 10 个学生成绩的数组。

任务 B:利用下标访问法,对上述数组进行正确的输入/输出操作,利用数组名作函数参数在函数之间传递数据,并实现学生成绩的输入/输出。

任务 C:设计“班级学生成绩管理系统”查找最高、最低和不及格成绩函数,优化查找最高、最低成绩函数,并进行正确的调用。

任务 D:模仿任务 10 初步设计“学生通讯录”中相关数组并对数组元素的访问。

## 8.2 任务 11:初步完善学生成绩排序

本任务初步完善按升序排列学生成绩函数 AsceSort()和按降序排列学生成绩函数 DropSort()。在任务 10 基础上,做如下修改。

(1) 函数声明修改为:

```
void AsceSort(float score[],int stusize); //按升序排列学生成绩函数声明
void DropSort(float score[],int stusize); //按降序排列学生成绩函数声明
```

(2) 主函数和相应的函数调用应做修改外,其他内容不变。想一想,这两个函数调用应当怎样修改?

(3) 函数定义修改为:

```
void AsceSort(float score[],int stusize) //按升序排列学生成绩函数
{
 int i,j;
 float temp;
 float temp_score[STUSIZE]; //定义一个新数组
 system("cls");
 for(i=0;i<stusize;i++) //给新数组赋值
 temp_score[i]=score[i];
 for(i=0;i<stusize-1;i++) //双重循环实现学生成绩按升序排列
 for(j=0;j<stusize-i-1;j++)
 if(temp_score[j]>temp_score[j+1])
 {
 temp=temp_score[j];
 temp_score[j]=temp_score[j+1];
 temp_score[j+1]=temp;
 }
}
```

```
 gotoxy(5,5);
 printf("升序排列结果:");
 for (i=0;i<stusize;i++)
 printf("%6.1f",temp_score[i]);
 gotoxy(20,10);
 printf("升序排列成功,按任意键返回上级菜单!");
 getch();
}

void DropSort(float score[],int stusize) //按降序排列学生成绩函数
{
 int i,j;
 float temp;
 float temp_score[STUSIZE];
 system("cls");
 for(i=0;i<stusize;i++)
 temp_score[i]=score[i];
 for(i=0;i<stusize-1;i++)
 for(j=0;j<stusize-i-1;j++)
 if(temp_score[j]<temp_score[j+1])
 {
 temp=temp_score[j];
 temp_score[j]=temp_score[j+1];
 temp_score[j+1]=temp;
 }
 gotoxy(5,5);
 printf("降序排列结果:");
 for(i=0;i<stusize;i++)
 printf("%6.1f",temp_score[i]);
 gotoxy(20,10);
 printf("降序排列成功,按任意键返回上级菜单!");
 getch();
}
```

在排序函数中,创建并生成了一个新的成绩数组(temp\_score),其目的是在排序的过程中,不影响原成绩数组的排列。

请读者模仿任务 10 中 AsceSort 函数和 DropSort 函数的优化方法,将升、降序排列函数优化合并成一个新的函数,并正确地调用它。

单元能力训练任务分别如下。

任务 A:用“冒泡排序法”对含有 10 个学生成绩数据的数组进行升序和降序排列,并

将升、降序操作优化成一个新函数。

任务 B: 正确设计计算学生平均成绩和总成绩函数, 并进行正确调用。

任务 C: 正确掌握多个学生姓名的输入、输出方法。

任务 D: 模仿任务 11 对“学生通讯录”中相关数据进行排序。

## 8.3 必备知识与理论

### 8.3.1 数组概述

在前面各章所使用到的数据都属于基本数据类型(整型、实型、字符型), C 语言除了提供基本数据类型外, 还提供了构造类型的方法, 它们是数组类型、结构体类型、共同体类型等。构造数据类型是由基本数据类型按照一定的规则组成, 所以也称为“导出类型”。

下面首先简单介绍几个基本概念。

(1) 数组: 若干个具有相同数据类型的数据的有序集合。

(2) 数组元素: 数组中的元素。数组中的每一个数组元素具有相同的数据类型、名称, 不同的下标, 可以作为单个变量使用, 所以又称为下标变量。在定义一个数组后, 在内存中使用一片连续的空间依次存放数组的各个元素。

(3) 数组的下标: 数组元素位置的一个索引或指示。

(4) 数组的维数: 数组元素下标的个数。根据数组的维数可以将数组分为一维、二维、多维数组。这里主要介绍一维数组, 二维数组将在扩展知识与理论一节中介绍。

### 8.3.2 一维数组的定义及其应用

前面已经讲到, 数组是一组有序数据的集合, 数组中每一个元素的数据类型相同。用数组名和下标来唯一确定数组中的每一个元素。

只有一个下标的数组称为一维数组。一维数组中的各个数组元素是排成一行的一组下标变量, 用一个统一的数组名来标识, 数组元素用一个下标来指示其在数组中的位置。一维数组通常用单重循环来对数组元素进行处理。

#### 1. 一维数组的定义

##### 1) 定义

一维数组的定义格式为:

|                     |
|---------------------|
| 类型标识符 数组名[整型常量表达式]; |
|---------------------|

例如: `int array[10];`

含义: 定义了一个数组, 数组名为 `array`, 有 10 个元素, 元素的类型均为整型。这 10 个元素名分别是: `array[0]`、`array[1]`、`...`、`array[9]`。

## 2) 数组元素表示方法

一维数组元素表示格式为：

数组名[下标表达式]

数组元素相当于单个变量,它的属性与变量相同。

## 3) 说明

(1) 数组名:按标识符规则命名。上例中,array 就是数组名。

(2) 整型常量表达式:表示数组元素个数(数组的长度)。可以是整型常量或符号常量,不允许用变量,也不允许用实型常量。例如:

```
int n;
scanf("%d",&n);
int array[n];
```

上例中,n 是一个变量,在程序运行的过程中可以改变其值,C 语言是不允许用变量来确定数组大小的,因为在编译时,C 编译器不能根据已知数组大小分配内存空间。

(3) 整型常量表达式在说明数组元素个数的同时也确定了数组元素下标的范围,下标从 0 开始至整型常量表达式-1(注意,不是 1 至整型常量表达式)。使用数组元素时不能超过整型常量表达式-1 的范围,这种现象称为有界性。C 语言不检查数组下标是否越界,是否越界需要程序员自己控制,一旦越界,结果难以预料。

(4) 数组名后是用方括号,不能用圆括号。下面的用法错误:

```
int array(10);
```

(5) 类型标识符:指的是数据元素的类型,可以是基本数据类型,也可以是构造数据类型。本例数组元素是整型,每个元素占 4 个字节,因为有 10 个数组元素,所以共占用了 40 字节。C 编译程序为其分配了连续的内存空间,如图 8.1 所示。

(6) C 语言还规定,数组名代表数组在内存中的首地址,即 array 和 &array[0] 相当。

由此可以归纳出数组特性三要素如下:

- ① 数组元素类型相同;
- ② 数组长度固定;
- ③ 数组占用连续的内存空间。

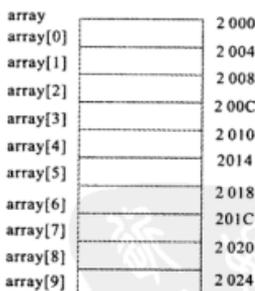


图 8.1 一维数组在内存中开辟空间示意图

## 2. 一维数组的初始化

一维数组初始化格式为:

类型标识符 数组名[整型常量表达式]={初值表};

初始化:在定义数组的同时指定初始值,编译器把初值赋予数组元素。

一维数组有下列几种初始化形式。

(1) 一般初始化。例如：

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

其结果是给每一个数组元素都指定了初值。

(2) 部分元素初始化,其余元素均为零。

例如: `int a[10]={7,8,9,67,54};`

仅给前 5 个元素赋初值,后 5 个元素初值为 0。

① 部分元素初始化时,编译器自动将没有初始化的元素初始化为 0。

② 数值型数组如果只定义不初始化,编译器不为数组自动指定初始值,即初值为随机值。

(3) 全部元素均初始化为 1。例如：

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

不允许简写为 `int a[10]={1};` 或 `int b[10]={1*10};`

想一想,上述定义 a 和 b 数组后,初始化结果是什么?

(4) 如果全部元素均指定初值,定义中可省略元素的个数。

例如: `int a[5]={9,28,3,4,5};`

可以写为: `int a[]={9,28,3,4,5};`

### 3. 一维数组元素引用

C 语言规定,不能整体引用数值数组,只能逐个引用数组元素。

#### 1) 下标法引用数组元素

引用格式为：

数组名[下标表达式]

例如: `a[0]=a[5]+a[7]-a[2*3]`

“下标表达式”可以是任何非负整型数据。

#### 2) 说明

(1) 一个数组元素,实质上就是一个变量,它具有与相同类型单个变量一样的属性,可以对它进行赋值和参与各种运算。

(2) 在 C 语言中,数值型数组作为一个整体,不能参加运算,只能对单个的元素进行处理。

**【例 8.1】** 键盘输入 10 个数给数组 a,然后逆序输出。

```
1 #include <stdio.h>
2 void main()
3 {
4 int i,a[10];
5 for(i=0;i<=9;i++)
```

```

6 scanf("%d",&a[i]); //数组元素取地址
7 for(i=9;i>=0; i--)
8 printf("%4d ",a[i]); //下标引用方法
9 }

```

程序运行结果:

```

0 1 2 3 4 5 6 7 8 9<回车>
9 8 7 6 5 4 3 2 1 0

```

**【例 8.2】** 输入 10 个 0 到 100 的随机整数到指定的数组中。

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 void main()
4 {
5 int i,a[10]={0};
6 srand(50); //初始化随机数序列*/
7 for(i=0;i<10;i++)
8 {
9 a[i]=rand()%100; //产生 100 以内的随机整数*/
10 }
11 for(i=0;i<10;i++)
12 printf("%4d ",a[i]);
13 }

```

程序运行结果:

```
1 51 10 54 96 25 48 94 30 21
```

注意:不同的机器可能输出结果不一样。

### 8.3.3 数组作函数参数

数组作函数参数有两种形式:一种是把数组元素(又称下标变量)作为实参使用;另一种是把数组名作为函数的形参或实参使用。

#### 1. 数组元素作函数参数

数组元素就是下标变量,它与普通变量并无区别。数组元素只能用作函数实参,其用法与普通变量完全相同:在发生函数调用时,把数组元素的值传送给形参,实现单向值传递。

**【例 8.3】** 写一函数,统计字符串中英文字母的个数。

```

1 int isalp(char c)
2 {
3 if(c>='a'&&c<='z' || c>='A'&&c<='Z')
4 return 1;
5 else

```

```

6 return 0;
7 }
9
10 #include <stdio.h>
11 void main()
12 {
13 int i,num=0;
14 char str[255];
15 printf("Input a string: ");
16 gets(str);
17 for(i=0;str[i]!='\0';i++)
18 if(isalp(str[i]))
19 num++;
20 printf("num=%d\n",num);
21 }

```

程序运行结果：

```

One World One Dream<回车>
num=16

```

## 2. 数组名作函数参数

数组名作函数参数时，既可以作形参，也可以作实参。

数组名作函数参数时，要求形参和相对应的实参都必须是类型相同的数组（或指向数组的指针变量），用一维数组作形参时，可以不限定数组元素的个数。

**【例 8.4】** 随机产生 0~100 之间的 10 个数，并求其和。要求产生 10 个随机数和求和运算分别写在不同的函数中，主函数显示这 10 个数和它们的和。

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 void random(int array[],int n); //函数声明
4 int count(int array[],int n); //函数声明
5 void main()
6 {
7 int i,a[10]={0};
8 int sum;
9 random(a,10); //调用求 10 个随机数函数
10 for(i=0;i<10;i++)
11 printf("%4d ",a[i]);
12 printf("\n");
13 sum=count(a,10); //调用求和函数
14 printf("sum=%d\n",sum);

```

```

15 }
16
17 void random(int array[],int n) //求随机数函数
18 {
19 int i;
20 srand(50); /*初始化随机数序列*/
21 for(i=0;i<n;i++)
22 {
23 array[i]=rand()%100; /*产生 100 以内的随机整数*/
24 }
25 }
26
27 int count(int array[],int n) //求和函数
28 {
29 int i,sum=0;
30 for(i=0;i<n;i++)
31 sum+=array[i];
32 return sum;
33 }

```

程序运行结果：

```

1 51 10 54 96 25 48 94 30 21
sum=430

```

在这个例子中,没有限定形参 array 数组的元素个数,有关元素个数的信息是通过形参表中另一个参数 n 传递的,这样做的好处就是提高了函数的通用性,提高阅读性。

**注意:**即使限定了数组参数元素个数,也没有什么意义,因为 C 语言并不因此而进行下标越界检查。

### 3. 说明

(1) 用数组元素作实参时,只要数组类型和函数的形参类型一致即可,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。

(2) 用数组名作函数参数,应该在调用函数和被调用函数中分别定义数组,且数据类型必须一致,否则结果将出错。例如,在例 8.4 中,形参数组为 array[ ],实参数组为 a[ ],它们的数据类型相同。数组名作函数参数,传递的是数组的首地址。

### 4. 一维数组应用实例

**【例 8.5】** 随机产生 0~100 之间的 10 个数,并用“冒泡法”对 10 个数排序(由小到大)。要求排序算法用函数来实现,主函数中输出排序前后结果。

冒泡法的基本思想:通过相邻两个数之间的比较和交换,使排序码(数值)较小的数逐渐从底部移向顶部,排序码较大的数逐渐从顶部移向底部,就像水底的气泡一样逐渐向上

冒，因此而得名。

“冒泡法”算法：以六个数9、8、5、4、2、0为例，两数之间的比较如图8.2所示。

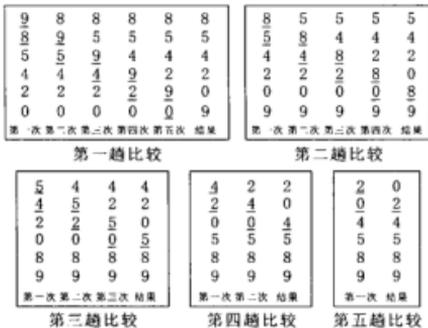


图 8.2 冒泡法两数之间的比较示意图

第1趟比较，有6个数未排好序，两两比较5次；

第2趟比较，剩5个数未排好序，两两比较4次；

第3趟比较，剩4个数未排好序，两两比较3次；

第4趟比较，剩3个数未排好序，两两比较2次；

第5趟比较，剩2个数未排好序，两两比较1次。

算法结论：对于n个数的排序，需进行n-1趟比较，第j趟比较需进行n-i-1次比较（由于j从0开始编号）。

冒泡法排序流程图如图8.3所示（下面以N-S图表示流程）。

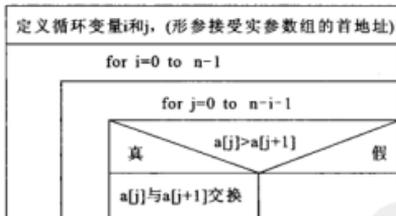


图 8.3 冒泡法排序 N-S 图

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 void random(int array[],int n); //函数声明
4 void AsceSort(int array[],int n); //函数声明
5 void main()
6 {
7 int i,a[10]={0};

```

```

8 random(a,10); //调用求 10 个随机数函数
9 printf("排序前:\n");
10 for(i=0;i<10;i++)
11 printf("%4d ",a[i]);
12 printf("\n");
13 AsceSort(a,10); //调用排序函数
14 printf("排序后:\n");
15 for(i=0;i<10;i++)
16 printf("%4d ",a[i]);
17 printf("\n");
18 }
19
20 void random(int array[],int n) //求随机数函数
21 {
22 与例 8.4 相同
23 }
24
25 void AsceSort(int array[],int n) //按升序排列
26 {
27 int i,j,temp;
28 for(i=0;i<n-1;i++)
29 {
30 for(j=0;j<n-i-1;j++)
31 {
32 if(array[j]>array[j+1])
33 {
34 temp=array[j];
35 array[j]=array[j+1];
36 array[j+1]=temp;
37 }
38 }
39 }
40 }

```

程序运行结果:

排序前:

1 51 10 54 96 25 48 94 30 21

排序后:

1 10 21 25 30 48 51 54 94 96



### 8.3.4 字符数组的定义及其应用

通过前面的学习可以知道,字符分为字符常量和字符变量,我们还知道 C 语言有字符串常量,那么有没有字符串变量呢? C 语言中没有字符串变量类型,字符串变量是借助于字符数组来实现的。

#### 1. 字符数组的定义

存放字符数据的数组称为字符数组,在这种数组中每一个元素存放一个字符。

##### 1) 定义格式

```
char 数组名[常量表达式];
```

例如:char c[10]; /\* 定义 c 为字符数组 \*/

可以给每一个数组元素赋值,例如:

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

赋值以后字符数组在内存中存储状态如图 8.4 所示。

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

图 8.4 字符数组在内存中存储状态示意图

##### 2) 说明

(1) 字符数组的定义格式与前面讲的一维、二维数组的定义格式相似,只是类型标识符一定为 char。

(2) 一般将用 int 或 float 定义的数组称为数值型数组,字符型数组除了具有数值型数组的属性外,还具有自己特殊的特性。

#### 2. 字符数组的初始化

字符数组的初始化有下面多种方法。

(1) 逐个元素赋值初始化。例如:

```
char ch[10]={'C','h','i','n','a'};
```

如果初始化数据小于数组长度,多余元素自动赋为'\0'(字符 0)。如图 8.5 所示。

(2) 还可以用字符串常量给字符数组赋初值。例如:

```
char ch[6]="china";
```

(3) 指定初值时,若未指定数组长度,则长度等于初值个数。例如:

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

数组长度为 10,包括两个空格。

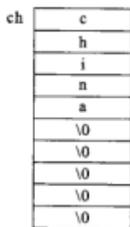
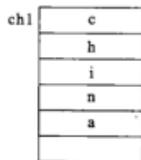
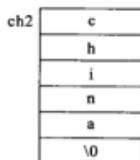


图 8.5 字符数组的初始化



(a)



(b)

图 8.6 不同初始化方法占用内存区别示意图

(4) 不指定数组长度, 逐个元素初始化和字符串常量初始化在内存中的存储情况是不一样的。例如:

`char ch1[]={'c','h','i','n','a'};`在内存中占 5 个字节。(如图 8.6(a)所示)

`char ch2[]="china";`在内存中占 6 个字节。(如图 8.6(b)所示)

因为字符串常量末尾自动带上一个 '\0', 所以用字符串常量对字符数组初始化, 比用逐个元素初始化(与字符串字符个数相同)字符数组多占用一个字节的内存空间。

### 3. 字符数组的引用

上面定义的字符数组都是一维数组, 因此它们具有一维数组的属性。引用一个数组元素, 将得到一个字符, 字符数组的引用可以像一维数组一样, 用一重循环实现对字符数组元素的操作。

**【例 8.6】** 用字符串常量初始化一个字符数组, 然后输出这个字符串。

```

1 #include <stdio.h>
2 void main()
3 {
4 char c[20]="I am boy";
5 int i=0;
6 while(c[i]!='\0')
7 {
8 printf("%c",c[i]);
9 i++;
10 }
11 printf("\n");
12 }

```

程序运行结果:

```
I am boy
```

**【例 8.7】** 输出一个钻石图形。

```

1 #include <stdio.h>
2 void main()

```



```

3 {
4 char diamond[][5]={{ ' ', ' ', ' ', '* ' },
5 { ' ', '* ', ' ', ' ', '* ' },
6 { '* ', ' ', ' ', ' ', ' ', '* ' },
7 { ' ', '* ', ' ', ' ', '* ' },
8 { ' ', ' ', ' ', '* ' }};
9 int i,j;
10 for(i=0;i<5;i++)
11 {
12 for(j=0;j<5;j++)
13 printf("%c",diamond[i][j]);
14 printf("\n");
15 }
16 }

```

程序运行结果:

```

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

```

**【例 8.8】** 从键盘输入一字符串,并将其输出。

```

1 #include <stdio.h>
2 void main()
3 {
4 char ch[81];
5 printf("请输入字符串(不包括空格字符):");
6 scanf("%s",ch); /*输入字符串*/
7 printf("%s\n",ch); /*输出字符串*/
8 }

```

程序运行结果:

```

请输入字符串(不包括空格字符):Beijing2008奥运会<回车>
Beijing2008奥运会

```

说明:

(1) C语言中,数组名代表该数组的起始地址,因此,scanf()函数中不需要地址运算符&。

例如:char str[13];

```

scanf("%s",str); /*正确*/
scanf("%s",&str); /*错误*/

```

(2) “%s”格式输出时,printf()函数的输出项是字符数组名,而不是数组元素名。另外,即使数组长度大于字符串长度,遇‘\0’也结束。

例如:char c[10]="China";

```
printf("%s",c); /*只输出5个字符,而不是10个字符*/
```

(3) scanf()函数输入一个字符串时,以空格或回车作为字符串的结束标志。

例如:scanf("%s",ch);

```
printf("%s",ch);
```

如果输入的是:How are you,输出结果为:How。后面的 are you 不能送入到 ch 中。

(4) 不能采用赋值语句将一个字符串直接赋给一个数组。

例如:char c[10]; c="good"; /\*错误\*/

#### 4. 字符串处理函数

为了方便用户操作字符串,C语言的库函数为我们提供了大量字符串处理函数。下面介绍常用的几个函数。希望能举一反三,掌握C语言库函数的使用方法。

在调用字符串处理函数时,在使用库函数之前必须设置一个相关的文件包含预处理命令,即:#include <string.h>。

##### 1) 字符串输出函数(puts())

格式为:

```
puts(字符数组);
```

输出字符串(以‘\0’结尾的字符序列)。

例如:char c[6]="China";

```
puts(c); /*不需要格式控制符,且自动换行*/
```

上述函数调用与下面的函数调用等价。

```
printf("%s\n",c); /*需要格式控制符%s和\n*/
```

##### 2) 字符串输入函数(gets())

格式为:

```
gets(字符数组);
```

输入字符串到相应的字符数组中。

例如:char str[12];

```
gets(str);
```

如果输入“How are you”,gets函数可以将其全部送入以str为首址的内存中,即gets函数以回车作为结束输入的标志,而不包括空格。

注意:gets()、puts()一次只能输入输出一个字符串。而scanf()、printf()一次可以输入/输出几个字符串。

##### 3) 字符串连接函数(strcat())

格式为:



```
strcat(字符数组 1, 字符数组 2);
```

功能:把“字符数组 2”连接到“字符数组 1”的后面。从字符数组 1 第一个‘\0’(字符串结束标志)处开始连接,结果存放在字符数组 1 中。

```
例如:char ch1[20]="computer is";
 char ch2[10]="good! ";
 printf("%s",strcat(ch1,ch2));
```

输出结果为:computer is good!

两字符串连接时内存变化示意如图 8.7 所示。

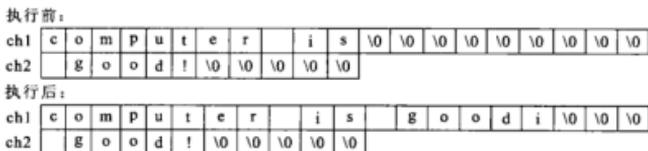


图 8.7 两字符串连接时内存变化示意图

执行后可以看出 ch1 数组发生了变化,而 ch2 数组不变。

**注意:**字符数组 1 要有足够的空间,以确保连接字符串后不越界,字符数组 2 可以是字符数组名,字符串常量或指向字符串的字符指针(地址)。

#### 4) 字符串拷贝函数(strcpy())

格式为:

```
strcpy(字符数组 1, 字符数组 2);
```

功能:将“字符数组 2”为首地址的字符串复制到“字符数组 1”为首地址的字符数组中。即把“字符数组 2”的值拷贝到“字符数组 1”中。

```
例如:char str1[10],str2[]="china";
 strcpy(str1,str2);
```

**注意:**str1 一般为字符数组,要有足够的空间,以确保复制字符串后不越界;str2 可以是字符数组名,也可以是字符串常量或指向字符串的字符指针(地址)。

字符串(字符数组)之间不能用赋值符赋值,即不能使用赋值运算符复制字符串。str1=str2;是错误的。通过此函数,可以间接达到赋值的效果。

#### 5) 字符串比较函数(strcmp())

格式为:

```
strcmp(字符数组 1, 字符数组 2);
```

功能:比较“字符数组 1”和“字符数组 2”,例如:

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

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

比较规则:逐个字符比较其 ASCII 码,直到遇到不同字符或'\0',比较结果是该函数的返回值。

- ① 字符数组 1==字符数组 2,strcmp()返回值为 0;
- ② 字符数组 1>字符数组 2,strcmp()返回值为 一正整数;
- ③ 字符数组 1<字符数组 2,strcmp()返回值为 一负整数。

注意:字符串只能用 strcmp 函数比较,不能用关系运算符“==”比较。

例如:if (strcmp(str1,str2)==0) printf("yes");

而 if (str1 == str2) printf("yes");是错误的。

#### 6) 求字符串的长度函数(strlen())

格式为:

```
strlen(字符数组);
```

功能:统计字符数组中字符串的长度(不包括字符串结束标志),并将其作为函数值返回。

例如:char str[10]="Beijing";  
printf("%d",strlen(str));

输出结果为 7 而不是 8。

#### 7) 大写字母转变为小写字母函数(strlwr())

格式为:

```
strlwr(字符数组);
```

功能:将字符数组中大写字母全部转变成小写字母。

例如: char str[10]="HELLO";  
printf("%s",strlwr(str));

输出结果为:hello

#### 8) 小写字母转变为大写字母函数(strupr())

格式为:

```
strupr(字符数组);
```

功能:将字符数组中小写字母全部转变成大写字母。

例如: char str[10]="hello";  
printf("%s",strupr(str));

输出结果为:HELLO

### 5. 字符数组应用实例

**【例 8.9】** 输入/输出学生姓名,姓名中可以包括空格。



```

1 #include <stdio.h>
2 #include <string.h>
3 void main()
4 {
5 char name[20];
6 printf("请输入姓名(姓名中可以包含空格):");
7 gets(name);
8 printf("你输入的姓名是:%s\n",name);
9 }

```

程序运行结果:

请输入姓名(姓名中可以包含空格):张 三<回车>

你输入的姓名是:张 三。

**【例 8.10】** 输入一行字符,统计其中有多少个单词(单词间以空格分隔)。

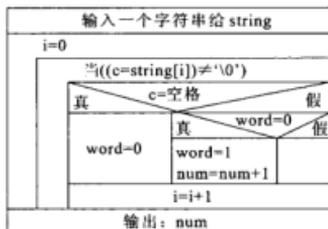


图 8.8 例 8.10 程序流程 N-S 图

解题思路:单词的数目由空格出现的次数决定(连续出现的空格记为出现一次,一行开头的空格不算)。应逐个检测每一个字符是否为空格。如果测出某一个字符为非空格,则表示“新的单词开始了”,此时用 num 表示单词数(初值为 0)。word=0 表示前一字符为空格,word=1 表示前一字符不是空格,word 初值为 0。如果前一字符是空格,当前字符不是空格,说明出现新单词, num 加 1。

程序流程图如图 8.8 所示。

```

1 #include "stdio.h" /* gets() 函数在该头文件定义 */
2 void main()
3 {
4 char string[81];
5 int i, num=0, word=0;
6 char c;
7 gets(string);
8 for(i=0; (c=string[i]) != '\0'; i++)
9 if(c==' ') word=0;
10 else if(word==0)
11 {
12 word=1;
13 num++;
14 }
15 printf("There are %d words in the line.\n",num);
16 }

```

程序运行结果:

```
I am boy<回车>
```

```
There are 3 words in the line.
```

想一想:能否用 scanf("%s",string);来代替第 7 行?为什么?

## 8.4 扩展知识与理论

### 8.4.1 二维数组的定义及其应用

二维数组是数组元素行和列两方面受限的数组。二维数组也用统一的数组名来标识,它有两个下标,第一个下标表示行,第二个下标表示列。

#### 1. 二维数组的定义

##### 1) 定义格式

类型标识符 数组名[常量表达式 1][常量表达式 2];

例如: `int a[3][4];` //a 为 3×4 (3 行 4 列) 的数组

`float b[5][10];` //b 为 5×10 (5 行 10 列) 的数组

二维数组 `a[3][4]` 可以理解为:有三个一维数组元素 `a[0]`、`a[1]`、`a[2]`,而每个一维数组元素又是一个包含 4 个元素的一维数组,如图 8.9 所示。

二维数组的逻辑结构是二维的,但物理结构(存储结构)却是一维的。

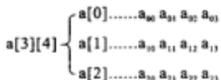


图 8.9 二维数组元素又是一个一维数组示意图

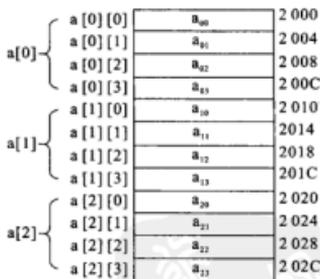


图 8.10 按行优先存储二维数组示意图

二维数组元素在内存中的存放顺序:C 语言规定,二维数组“按行优先”的方法存放,即先顺序存放第一行的元素,再存放第二行的元素,……(第二维下标变化较快,第一维下标变化较慢)。如图 8.10 所示。

#### 2) 说明

(1) 二维数组中的每个数组元素都有两个下标,且必须分别放在单独的“[]”内。

`a[3,4]`是错误的。

(2) 二维数组定义中的第1个下标表示该数组具有的行数,第2个下标表示该数组具有的列数,两个下标之积是该数组具有的数组元素个数。

(3) 行下标的取值范围为 $0 \sim$ 常量表达式 $1-1$ ,列下标的取值范围为 $0 \sim$ 常量表达式 $2-1$ 。

(4) 二维数组中的每个数组元素的数据类型均相同。二维数组在内存中的存放规律是“按行优先”。

(5) 二维数组可以看作是数组元素为一维数组的数组。

## 2. 二维数组的初始化

二维数组初始化有多种方法。

(1) 分行赋值,例如:

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

(2) 全部数据写在一个大括号内,例如:

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

(3) 部分元素赋值,例如:

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

仅对`a[0][0]`、`a[1][0]`、`a[2][0]`赋值,其余元素均初始化0。

(4) 如果对全部元素赋初值,则第一维的长度可以不指定,但必须指定第二维的长度。例如:

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

与下面定义等价:`int a[3][4]={1,2,3,4,5,6,7,8,9,10,11,12};`

## 3. 二维数组元素的引用

二维数组元素的下标法引用与一维数组元素的引用相似,只是二维数组元素要同时指明行下标与列下标。

### 1) 下标法引用二维数组

引用格式为:

数组名[行下标表达式][列下标表达式]

例如:`float a[3][3];`

其中,9个二维数组元素分别为:

```
a[0][0],a[0][1],a[0][2]
```

```
a[1][0],a[1][1],a[1][2]
```

```
a[2][0],a[2][1],a[2][2]
```

“行下标表达式”和“列下标表达式”可以是任何非负整型数据。

### 2) 说明

(1) 二维数组元素与相同类型的变量属性相同,它也称为下标变量。



(2) 用下标法访问二维数组,一般采用二重循环的方法来处理。

**【例 8.11】** 设计一个二维数组 `float a[3][3]`,并初始化,然后按矩阵方式输出。

解题思路:依题意,输出 3 行 3 列矩阵,行列都要变化,用二重循环来实现。第一个二维数组元素是 `a[0][0]`,最后一个二维数组元素是 `a[2][2]`,任意一个二维数组元素可表示为 `a[i][j]`,其中下标 `i` 的取值范围为  $0 \leq i \leq 2$ ,下标 `j` 的取值范围为  $0 \leq j \leq 2$ 。为了得到矩阵结果,在一行输出结果后,应输出一个回车换行。

程序流程图如图 8.11 所示。

```

1 #include <stdio.h>
2 void main()
3 {
4 float a[3][3]={{1.1f,1.2f,1.3f},{2.1f,2.2f,2.3f},{3.1f,3.2f,3.3f}};
5 int i,j;
6 for(i=0;i<3;i++)
7 {
8 for(j=0;j<3;j++)
9 printf("%6.1f",a[i][j]);
10 printf("\n");
11 }
12 }

```

程序运行结果:

```

1.1 1.2 1.3
2.1 2.2 2.3
3.1 3.2 3.3

```

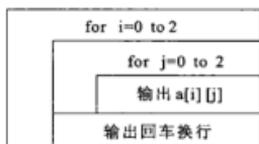


图 8.11 例 8.11 程序流程 N-S 图

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

图 8.12 a、b 两矩阵转置前后示意图

**【例 8.12】** 将一个二维数组行和列交换(转置),存到另一个二维数组中。如图 8.12 所示。

解题思路:只要将两数组的行列下标交换赋值即可,  $b[j][i] = a[i][j]$ 。

```

1 #include <stdio.h>
2 void main()
3 {
4 int a[2][3]={{1,2,3},{4,5,6}};
5 int b[3][2], i,j;
6 printf("array a:\n");

```

```

7 for(i=0;i<2;i++) //0~1行
8 {
9 for(j=0;j<3;j++) //0~2列
10 {
11 printf("%5d",a[i][j]);
12 b[j][i]=a[i][j]; //行、列交换
13 }
14 printf("\n");//输出一行后换行
15 }
16 printf("array b:\n");
17 for(i=0;i<3;i++)
18 {
19 for(j=0;j<2;j++)
20 printf("%5d",b[i][j]);
21 printf("\n");//输出一行后换行
22 }
23 }

```

程序运行结果：

```

array a:
 1 2 3
 4 5 6

array b:
 1 4
 2 5
 3 6

```

**【例 8.13】** 有一个  $3 \times 4$  的矩阵，要求编程以求出其中值最大的那个元素的值及其所在的行号和列号。

解题思路：首先假定第一个元素  $a[0][0]$  作为临时最大值  $\max$ ，然后把临时最大值  $\max$  与后续每一个元素  $a[i][j]$  进行比较，若  $a[i][j] > \max$ ，把  $a[i][j]$  作为新的临时最大值，并记录其下标  $i$  和  $j$ 。当全部元素比较完后， $\max$  便是整个矩阵全部元素中的最大值。

程序流程图如图 8.13 所示。

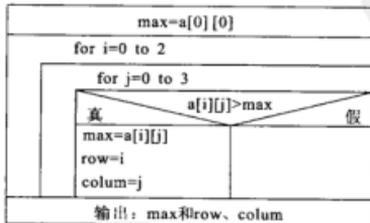


图 8.13 例 8.13 程序流程 N-S 图

```

1 #include <stdio.h>
2 void main()
3 {
4 int i,j,row=0,col=0,max;
5 int a[3][4]={{1,2,3,4},{9,8,7,6},{-10,10,-5,2}};
6 max=a[0][0];
7 for(i=0;i<=2;i++) /*用两重循环遍历全部元素*/
8 for(j=0;j<=3;j++)
9 if(a[i][j]>max)
10 {
11 max=a[i][j];
12 row=i;
13 col=j;
14 }
15 printf("max=%d, row=%d, col=%d\n",max,row,col);
16 }

```

程序运行结果：

```
max=10,row=2,col=1
```

二维数组的数组名也可以作为函数参数进行传递,它传递的是二维数组的首地址。

**【例 8.14】** 将例 8.13 中的查找最大值和行列号操作放在函数中,主函数调用该函数,并输出最大值及其所在行、列号。

解题思路:由于 return 语句只能返回一个值,因此,设计两个全局变量来得到行、列号,用 return 返回最大值。

```

1 #include <stdio.h>
2 int row=0,col=0; //全局变量
3 int count(int array[][4],int n,int m);
4 void main()
5 {
6 int max;
7 int a[3][4]={{1,2,3,4},{9,8,7,6},{-10,10,-5,2}};
8 max=count(a,3,4);
9 printf("max=%d, row=%d, col=%d\n",max,row,col);
10 }
11
12 int count(int array[][4],int n,int m) //行号可省,列号不能省
13 {
14 int i,j;
15 int max;
16 max=array[0][0];

```

```
17 for(i=0;i<n;i++) /*用双重循环遍历全部元素*/
18 for(j=0;j<m;j++)
19 if(array[i][j]>max)
20 {
21 max=array[i][j];
22 row=i;
23 colum=j;
24 }
25 return max;
26 }
```

程序运行结果:

```
max=10,row=2,colum=1
```

## 8.4.2 常见错误及处理方法

常见错误 1:用变量或实型常量定义数组。

C 语言规定,数组开辟的空间不能随程序的运行而改变大小,所以不能用变量定义数组空间的大小;这个空间是以字节为最小单位分配的,字节数不可能出现小数,因此数组长度也不能用实型常量来定义。

常见错误 2:scanf("%c",a[i]);或 scanf("%s",&a);。

第一条语句是给一个字符数组元素赋值,数组元素相当于变量,所以 a[i] 前面应加上取地址符“&”。第二条语句由于使用了 %s 格式控制符,可以对字符数组进行整体的输入,由于数组名就代表数组的首地址,因此,不必在数组名前再加取地址符。两条语句的正确形式应为:

```
scanf("%c",&a[i]);
scanf("%s",a);
```

常见错误 3:输入字符串的长度超过字符数组的长度,造成输出得不到预期的结果。

由于 C 语言不会自动检查数组下标越界错误,所以在实际应用中很容易出现下标越界的错误,因此要时刻注意在程序运行过程中下标是否越界。

常见错误 4:使用 strcpy 函数时出现被复制字符串与目标串颠倒。

使用 strcpy 函数时一定要注意,是将第二个实参复制给第一个实参,不能颠倒。

常见错误 5:用 %s 输入带空格的字符串。

使用 %s 格式控制符输入字符串时,它是以空格或回车作为结束输入的标志,一遇到空格或回车,系统将认为输入字符串结束,空格或回车后面的字符将不会送到字符数组中去。改进的方法是将输入函数 scanf() 换成 gets() 函数,用 gets() 函数输入的字符串中可以包括空格,但不包括回车符。

常见错误 6: int a[ ][ ]={ {1,2,3}, {3,2,1} };

在初始化二维数组时,两个下标常量表达式都省略了,这样就无法知道这个二维数组是几行几列的二维数组,系统在编译时将提示初始化错误,正确的初始化方法是行下标常量表达式可省,而列下标常量表达式不能省。

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

有了列数的信息,就能够计算出行数了。

## 8.5 深入训练

- 设计一个程序:输入 10 个整型数据到一个数组,然后逆序输出该数组。
- 利用冒泡排序法对十个从键盘输入的实型数据进行排序。
- 设计一程序,输入一个 4 行 4 列的行列式,显示这个二维数组组成的矩阵。
- 设计一程序,处理一个班的计算机成绩,要求采用一维数组来存储学生的成绩,计算平均成绩,打印每个学生的成绩和平均成绩。
- 编程实现五个学生姓名的输入/输出。要求:用二维字符数组存储姓名,输入姓名和输出姓名分别写在不同的函数中,主函数调用输入/输出姓名函数。
- 定义一个含有 30 个元素的整型数组,设计一个函数用于赋给该数组能被 3 整除的数,再设计一个函数,实现该数组按每 5 个元素计算平均值,并将平均值保存在一个新的数组中。
- 设计程序实现随意输入三个字符串,找出其中最大的字符串并输出。
- 打印“魔方阵”。所谓魔方阵是指它的每一行、每一列和对角线之和都相等。例如,三阶魔方阵为:

```
8 1 6
3 5 7
4 9 2
```

- 已知一行电文的按下列规则翻译成密码:即第 1 个字母变成第 26 个字母,第  $i$  个字母变成  $(26-i+1)$  个字母,非字母字符不变,要求编程实现原文变密码,密码变原文。

```
A->Z a->z
B->Y b->y
C->X c->x
... ...
```

## 习 题 8

### 8.1 填空题

- 已知数组  $a$  定义为  $\text{int } a[] = \{1, 2, 3, 4, 5\}$ ; 则数组  $a$  中各元素的值分别是 \_\_\_\_\_, 最小下标是 \_\_\_\_\_, 最大下标是 \_\_\_\_\_。
- 数组的三要素分别是 \_\_\_\_\_、\_\_\_\_\_ 和 \_\_\_\_\_。
- 已知数组定义为  $\text{double array}[50]$ , 则  $\text{array}$  的元素个数是 \_\_\_\_\_, 最小下标是 \_\_\_\_\_, 最大下标是 \_\_\_\_\_, 每个元素的类型是 \_\_\_\_\_。
- 已知数组  $A$  为一个有 14 个单元的长整型数组, 下面的语句试图从  $A[0]$  开始, 隔

一个输出一个 A 中的元素,请补充完整下面的语句:

```
for(_____; j< 14; _____) printf("% ld", _____);
```

(5) 已知数组 B 是一个有 15 个元素的整型数组,下面的语句是求这 15 个元素的平均值,并用 B0 来保存这个平均值,请补充完整下面的语句:

```
int j;int B0 _____;
for(_____; _____;j+) _____;
B0 _____ 15;
```

(6) 二维数组名为 d[m][n],其中 m 称为\_\_\_\_\_,n 称为\_\_\_\_\_。最大行下标为\_\_\_\_\_,最大列下标为\_\_\_\_\_。

(7) 在 C 语言中,二维数组元素在内存中的存放顺序是\_\_\_\_\_。

(8) 二维数组定义时,可以省略行下标,但必须指定\_\_\_\_\_。

(9) 在 C 语言中,判断字符数组是否结束的标记是\_\_\_\_\_。

(10) 在 C 语言中,数组下标的数据类型只能是\_\_\_\_\_。

(11) 已知数组 d 定义为 double d[4][11],则 d 是一个\_\_\_\_\_行\_\_\_\_\_列的二维数组,总共有\_\_\_\_\_个元素,最大行下标是\_\_\_\_\_,最大列下标是\_\_\_\_\_。

(12) 已知数组 d 定义为 int d[ ][3]={1,2,3},{5,6},{7};二维数组有\_\_\_\_\_行,按行列出各元素的值是\_\_\_\_\_。

(13) 已知数组 k 定义为 float k[4][5]={1,2,3,4,5,6,7,8,9,10,11,12},则 k 是一个有\_\_\_\_\_行\_\_\_\_\_列的二维数组,共有\_\_\_\_\_个元素,该数组各元素的值分别为\_\_\_\_\_。

(14) 执行下列语句后,数组 str2 中的字符串是\_\_\_\_\_。

```
char str1[]="ABCDE",str2[]="UVWXYZ";
for(j=0;str2[j]=str1[j];j++);
```

(15) 已知 s1,s2,s3 是三个有足够元素个数的字符串变量,其值分别为“aaa”、“bbb”、“ccc”,执行语句 strcat(s1, strcat(s2,s3))后,s1,s2 和 s3 的值分别是\_\_\_\_\_、\_\_\_\_\_、\_\_\_\_\_。

(16) 在 C 语言中数值型数组只能采用\_\_\_\_\_访问,字符型数组除了可以采用下标法访问外,还能采用\_\_\_\_\_访问。

**8.2 判断题**(判断下列叙述的正确性,正确的请打“√”,错误的请打“×”)

- (1) 说明语句 int a[7]={5,6,7};由于数组长度与初值个数不同,故该语句不正确。 ( )
- (2) 两个字符串所包含的字符个数相同时,才能比较字符串的大小。 ( )
- (3) 如果 a 是二维数组名,则 a+1 和 \*(a+1)的值相同。 ( )
- (4) int a[10]={}是正确的对一维数组 a 进行初始化的语句。 ( )
- (5) char a[]={0,1,2,3,4,5,6};不是正确的数组说明语句。 ( )
- (6) 调用 strlen("hello\ock\oby")的返回值为 12。 ( )

**8.3 选择题**

(1) 下面所列的正确的数组名标识是( )。

A. 1234

B. a1

C. a?

D. \ald

(2) 如定义了 `int A[10]`, 则该数组最后一个元素为( )。

- A. `A[10]`      B. `A[1]`      C. `A[0]`      D. `A[9]`

(3) 在 C 语言中, 引用数组元素时, 其数组下标的数据类型允许是( )。

- A. 整型常量      B. 整型表达式  
C. 整型常量或整型表达式      D. 任何类型的表达式

(4) 若有说明, `int a[10]`; 则对 `a` 数组元素的正确引用是( )。

- A. `a[10]`      B. `a[3.5]`      C. `a(5)`      D. `a[10-10]`

(5) (多选) 要定义一个 `int` 型一维数组 `array`, 并使其各元素具有初值 `9, 0, 3, 0, 0` 正确的定义语句有( )。

- A. `int array[] = {9, 0, 3};`      B. `int array[5] = {9, 0, 3};`  
C. `int array[5] = {9, 0, 3, 0};`      D. `int array[] = {9, 0, 3, 0, 0};`

(6) 下面关于数据的叙述, 正确的是( )。

- A. 数组元素的数据类型都相同  
B. 数组不经过定义也可以使用  
C. 同一数组, 允许有不同数据类型的数组元素  
D. 数组名等同于数组的第一个元素

(7) 对以下说明语句的正确理解是( )。

```
int a[10] = {6, 7, 8, 9, 10};
```

- A. 将 5 个初值依次赋给 `a[1]` 至 `a[5]`  
B. 将 5 个初值依次赋给 `a[0]` 至 `a[4]`  
C. 将 5 个初值依次赋给 `a[6]` 至 `a[10]`  
D. 因为数组长度与初值的个数不相同, 所以此语句不正确

(8) (多选) 要定义一个 `int` 型二维数组 `array`, 并使其各元素的值分别为:

```
2 3 0 0
3 0 0 0
0 0 0 0
1 2 3 4
```

则正确的定义语句是( )。

- A. `int array[] [4] = {{2, 3}, {3}, {0}, {1, 2, 3, 4}};`  
B. `int array[] [4] = {2, 3, 0, 0}, {3, 0, 0, 0}, {0, 0, 0, 0}, {1, 2, 3, 4}};`  
C. `int array[4] [4] = {{2, 3}, {3}, {}, {1, 2, 3, 4}};`  
D. `int array[4] [4] = {{2, 3}, {3}, {1, 2, 3, 4}};`



## 单元9 项目中指针的应用

### 能力目标

- 能正确定义指向不同类型数据的指针变量,能正确使用指针访问数据,能正确使用地址符和间接运算符。
- 能正确进行指针变量的运算。
- 能用指针变量作函数参数。
- 能用指针访问方法实现“班级学生成绩管理系统”中相关操作。

### 知识目标

- 理解指针与地址的概念和它们的区别。理解指针变量的定义,间接运算符的运算规则,理解指针在内存中是如何移动的。
- 理解指针变量作函数参数与变量作函数参数的区别。
- 理解指针在数组中移动的方法。

### 学习提示

地址与指针是C语言最为重要的内容之一,它是C语言的精华,它的主要优点是通过对地址与指针可以直接操作计算机的硬件(如内存),有效地提高程序的运行效率。学习C语言如果不能掌握地址与指针这个内容,就不能算真正掌握C语言。

### 9.1 任务12:用指针实现学生最高、最低等成绩查找

任务10给出的“班级学生成绩管理系统”查找最高分、最低分和不及格成绩的三个函数也可以用指针变量作参数来实现。下面给出的这三个函数,只将原函数中的数组形参修改成指针形参,函数按指针访问方式编写。

(1) 函数声明可修改成:

```
void SearchMax(float*,int); //查找最高分指针访问函数
```

```
void SearchMin(float*,int); //查找最低分指针访问函数
void NotElig(float*,int); //查找不合格学生成绩指针访问函数
```

(2) 函数调用可以不修改。

(3) 函数定义修改成:

```
void SearchMax(float*pscore,int stusize) //查找最高分指针访问函数
```

```
{
 float max=*pscore;
 int i,flag;
 system("cls");
 for(i=1;i<stusize;i++)
 {
 if(max<* (pscore+i))
 {
 max=* (pscore+i);
 flag=i;
 }
 }
 gotoxy(20,5);
 printf("成绩最高的是:%.1f\n",* (pscore+flag));
 gotoxy(20,10);
 printf("查找最高分成功,按任意键返回上级菜单!");
 getch();
}
```

```
void SearchMin(float*pscore,int stusize)
```

```
//查找最低分指针访问函数
```

```
{
 float min=*pscore;
 int i,flag;
 system("cls");
 for(i=1;i<stusize;i++)
 {
 if(min>* (pscore+i))
 {
 min=* (pscore+i);
 flag=i;
 }
 }
}
```



```

gotoxy(20,5);
printf("成绩最低的是:%.1f\n",*(pscore+flag));
gotoxy(20,10);
printf("查找最低分成功,按任意键返回上级菜单!");
getch();
}

```

```
void NotElig(float*pscore,int stusize)
```

//查找不合格学生成绩指针访问函数

```

{
 int i,flag=0;
 system("cls");
 gotoxy(20,5);
 printf("不合格成绩:");
 for(i=0;i<stusize;i++)
 {
 if(*(pscore+i)<60)
 {
 printf("%6.1f",*(pscore+i));
 flag=1;
 }
 }
 if(!flag)
 {
 gotoxy(35,5);
 printf("没有不合格成绩!");
 }
 gotoxy(20,10);
 printf("查找不合格成绩成功,按任意键返回上级菜单!");
 getch();
}

```

单元能力训练任务分别如下。

任务 A: 正确定义能指向学生成绩数组的指针变量, 正确运用指针的运算实现指针在学生成绩数组中的移动。

任务 B: 用指针变量实现“班级学生成绩管理系统”中学生成绩的查找。

任务 C: 用指向学生成绩数组的指针变量作函数参数, 实现查找学生最高分和最低分两个函数的优化。

任务 D: 模仿任务 12 实现对“学生通讯录”的相关操作。

## 9.2 任务 13:用指针实现学生成绩排序

(1) 函数声明可修改成:

```
void AsceSort(float*,int); //按升序排列指针访问函数
void DropSort(float*,int); //按降序排列指针访问函数
```

(2) 函数调用可以不修改。

(3) 函数定义修改成:

```
void AsceSort(float*pscore,int stusize) //按升序排列指针访问函数
```

```
{
 int i,j;
 float temp;
 float temp_score[STUSIZE],*pf;
 system("cls");
 pf=temp_score;
 for(i=0;i<stusize;i++,pscore++)
 temp_score[i]=*pscore;
 for(i=0;i<stusize- 1;i++)
 for(j=0;j<stusize- i- 1;j++)
 if(*(pf+j+1)<*(pf+j))
 {
 temp=*(pf+j);
 (pf+j)=(pf+j+1);
 *(pf+j+1)=temp;
 }
 gotoxy(5,5);
 printf("升序排列结果:");
 for(i=0;i<stusize;i++)
 printf("%6.1f",*(pf+i));
 gotoxy(20,10);
 printf("升序排列成功,按任意键返回上级菜单!");
 getch();
}
```

```
void DropSort(float*pscore,int stusize) //按降序排列指针访问函数
```

```
{
```



```

int i,j;
float temp;
float temp_score[STUSIZE],*pf;
system("cls");
pf=temp_score;
for(i=0;i<stusize;i++,pscore++)
 temp_score[i]=*pscore;
for(i=0;i<stusize- 1;i++)
 for(j=0;j<stusize- i- 1;j++)
 if(*(pf+j+1)> *(pf+j))
 {
 temp=*(pf+j);
 (pf+j)=(pf+j+1);
 *(pf+j+1)=temp;
 }
gotoxy(5,5);
printf("降序排列结果:");
for(i=0;i<stusize;i++)
 printf("%6.1f",*(pf+i));
gotoxy(20,10);
printf("降序排列成功,按任意键返回上级菜单!");
getch();
}

```

单元能力训练任务分别如下。

任务 A:应用指向学生成绩数组的指针变量作函数参数,实现学生成绩的升、降序排列。

任务 B:应用指向学生成绩数组的指针变量作函数参数,实现学生成绩的升、降序排列函数的优化。

任务 C:用指针实现输入/输出多个学生姓名的操作。

任务 D:模仿任务 13 实现“学生通讯录”排序的相关操作。

## 9.3 必备知识与理论

### 9.3.1 内存地址与数据指针的概念

所谓指针,是一种为方便对内存的直接访问而提供的一种语言机制。我们知道,如果要住旅馆,办完一定手续后,旅馆会提供住房号,根据住房号可以找到这个客房从而使用客房。同样,计算机内存空间是由顺序排列的以字节为单位的存储单元,将这些存储单元从 0 开始顺序编号,这些编号就构成了每个存储单元的地址,如图 9.1 所示。每个数据都

存放于从某个特定的地址开始的1个或若干个字节单元中,这个特定的地址就被认定为是该数据的存储地址,计算机对数据的存取都是通过这样的地址才得以实现的。

我们知道,数据是分类型的,而不同的数据类型在内存中占的空间大小是不同的,如整型占4个字节,双精度实型占8个字节等。现在的问题是每个字节都有地址,那么哪一个地址作为数据的地址呢?C语言规定,地址编号最小的地址作为数据(变量)的地址。例如: `int x;`,4个地址中最小的那一个就是变量 `x` 的地址,如图9.2所示; `float y;` 4个地址中最小的那一个是变量 `y` 的地址,如图9.3所示。

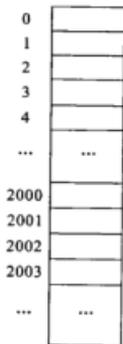


图 9.1 内存单元与地址示意图

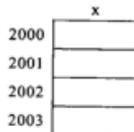


图 9.2 变量 `x` 的地址是 2000

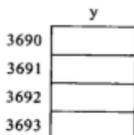


图 9.3 变量 `y` 的地址是 3690

在前面的学习中,我们对数据的存取基本上都是通过变量名进行的,没有直接与地址打交道。而每个变量名都与一个唯一的地址相对应,因此,我们对变量的访问实质上还是通过地址来进行数据的存取。由于编译系统所生成的代码能够自动地根据变量名与地址的对应关系完成相应的地址操作,因而一般情况下我们并不关心一个数据的具体存储地址,也不必为如何进行地址操作而操心。

但是,在某些类型的应用中,需要先“算出”数据的存储地址,然后再通过该地址间接地访问数据。在这种情况下,地址本身被作为数据处理的对象的一部分,成为一种特殊的数据。由于地址指明了数据存储的位置,因此形象地将地址称之为指针,该地址存放的数据也形象地称为“指针所指向的数据”。

指针与地址虽然有着密切的关系,但它们在概念上是有区别的,指针所标明的地址总是为保存特定的数据类型的数据而准备的,因此指针不但标明了数据的存储位置,而且还标明了该数据的类型,可以说指针是存储特定数据的地址。

现将地址与指针的概念归纳如下。

### 1. 变量定义的含义

由于C语言是一种强类型的程序设计语言,变量必须先定义然后才能使用。这意味着变量在使用之前,系统一定会为变量分配适当大小的内存空间,并建立变量名与变量地址相应的联系,以满足通过变量名访问数据的要求。

## 2. 存储单元地址(内存地址)

计算机在内存中存放数据时也需要内存地址,每一个内存单元都有一个唯一的内存地址与之对应,数据在内存中占用的单元大小可以不一样,有的大一些(如实型数据),有的小一些(如整型数据),但都是取最小地址为数据(变量)地址,也称为数据(变量)的内存地址。

## 3. 存储单元数据

存储单元数据即某一个内存单元中存放的数据。这些数据根据类型的不同占用空间大小也不同。要明白计算机中所描述的数据,不但有“值”的大小区别,而且还有存储“空间”大小的区别。

## 4. CPU 访问内存中数据的方式

CPU 对变量的访问是根据内存单元的地址来进行的。访问的方式如下。

### 1) 直接访问

含义:通过变量名直接存取数据。

特点:用户不需要知道数据具体的存放地址。

例如:`int x=3;printf("%d",x);`

### 2) 间接访问

含义:通过存放变量地址的变量(指针变量)来访问目标单元的值。

特点:用户需要事先知道数据存放的地址。

## 5. 指针的类型

指针的类型就是指针所指向的数据的类型。

指针的类型限定指针的用途,例如,一个 double 型指针只能用于指向 double 型数据。不限定类型的指针为无类型的指针或者说是 void 指针,可用于指向任何类型的数据。

## 9.3.2 指向变量的指针变量

### 1. 指针变量的定义和初始化

用来存放数据地址的变量称为指针变量。

#### 1) 定义格式

|                     |
|---------------------|
| 类型标识符 *变量名[=地址表达式]; |
|---------------------|

其中:类型标识符是指针变量所指向单元的值的数据类型。



“\*”是指针变量的定义符；

“变量名”命名规则同一般变量，但这个变量存放的是地址而不是数据值；

“地址表达式”是一个能表示成地址的表达式，一般在变量名前加取地址符“&”。

例如：`int x, * pointer; pointer=&x;`；则 `pointer` 指针变量存放的是变量 `x` 在内存中开辟的地址。

指针变量也是变量，在内存中也要占用存储空间。C语言规定：任何类型的指针变量占用空间的大小都是一样的，但不同的开发系统，实际占用字节数又是不同的，在 Visual C++ 系统中指针变量占用 4 个字节。

## 2) 指针变量的初始化

在定义指针变量的同时给指针变量赋地址值。

```
例如： int x=3;
 float y;
 int*pointer1=&x;
 float*pointer2=&y;
```

指针的指向过程如图 9.4 所示，`pointer1` 和 `pointer2` 为指针变量，`&x` 和 `&y` 为 `x` 和 `y` 变量的地址。

指针变量完成指向变量的过程是有次序的，首先，将变量的首地址赋给指针变量，然后，指针变量才指向变量。

指针变量使用之前，如果没有给指针变量赋值，即指针变量没有指向一个具体的地址，这样的指针称为空指针，空指针是“危险”的。因此，如果指针变量暂时不指向一个变量地址，请给指针变量赋 NULL 值（NULL 表示 0 内存单元地址）。

## 2. 指针变量的引用

指针变量运算符（间接访问运算符）“\*”是一个单目运算符，优先级比较高，用在指针变量名之前，获取该指针所指向的目标单元的值。

要注意“\*”号的不同意义，在定义变量时用“\*”号，表示定义了一个指针变量；在引用指针变量时用“\*”号，表示间接运算。另外，“\*”还是乘法运算符（双目运算符）。

**【例 9.1】** 指针与地址的应用。

```
1 # include <stdio.h>
2 void main ()
3 {
4 int a,b,*pointer_1,*pointer_2;
5 a=100,b=200;
6 pointer_1=&a;
7 pointer_2=&b;
```

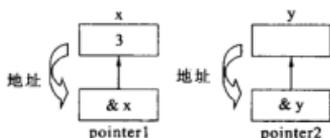


图 9.4 指针变量指向变量示意图

```

8 printf("%d,%d\n",a,*pointer_1); //用变量和指针变量输出数据值
9 printf("%d,%d\n",b,*pointer_2);
10 }

```

程序运行结果:

```

100 100
200 200

```

说明:

(1) 变量与指针变量如有下列定义:

```

DataType x,*p;
p=&x;

```

变量与指针变量的地址与数据值,一定满足于如图 9.5 所示在垂直方向的等价关系。

|         | 地址 | 数据值 |
|---------|----|-----|
| 变量 x:   | &x | x   |
| 指针变量 p: | p  | *p  |

图 9.5 变量与指针变量的地址与数据值等价关系示意图

(2) “&”和“\*”两个运算符的优先级是相同的,结合规律是右结合性。如:

若 point1=&a;,则 &\*point1 等价于 &a;, \* &a 等价于 a; (\*point1)++ 等价于 a++。

(3) 指针还可以指向数组或函数,分别称为数组指针和函数指针。

### 3. 指针变量的运算

#### 1) 算术运算

功能:对于地址的运算,只能进行整型数据的加、减运算。

规则:指针变量  $p \pm n$  表示将指针指向的当前位置向前或向后移动  $n$  个存储单元。

指针变量算术运算的过程:

$$p \pm n * \text{sizeof}(\text{类型})$$

注意: $p \pm n$  不是加(减) $n$  个字节,而是加(减) $n$  个数据单元。

#### 2) 关系运算

功能:用于标识目标变量在内存中的前后位置。

用法:  $\text{int } i, j;$

$\text{int } *p1 = \&i, *p2 = \&j;$

$p1 > p2$  用于标识变量  $i, j$  在内存的排列顺序。

#### 3) 赋值运算

功能:对指针变量的赋值运算,将改变指针变量所指向的地址,即改变指针的指向。

(1) 计算两地址间数据单元的个数(指针相减)。

同类型的两指针相减,其结果是一个整数,表示两地址之间可容纳的相应类型数据的个数,例如:

```
int n,m[12],*p1=&m[5],*p2=&m[10];
n=p2-p1;
```

两指针变量相减结果如图 9.6 所示。

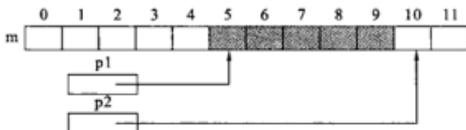


图 9.6 两指针变量相减结果示意图

相减结果是 5,它表示 5 个数据单元,共 20 个字节。

指针相减操作,一般只有高地址指针减低地址指针才有意义,指针相减操作不能用于指向函数的指针。

(2) 指针移动。

① 移动  $n$  个单位(加操作、减操作、加赋值操作、减赋值操作),这些操作可以引起指针向前或向后移动。

向前(低地址方向)移动指针,格式为:

|                 |                 |
|-----------------|-----------------|
| 指针变量=指针表达式- $n$ | //向前移动 $n$ 个单位数 |
| 或               |                 |
| 指针变量-= $n$      | //向前移动 $n$ 个单元数 |

向后(高地址方向)移动指针,格式为:

|                 |                 |
|-----------------|-----------------|
| 指针变量=指针表达式+ $n$ | //向后移动 $n$ 个单位数 |
| 或               |                 |
| 指针变量+= $n$      | //向后移动 $n$ 个单元数 |

例如,下面的语句进行了指针移动:

```
int m[12],*p1=&m[6],*p2=&m[8],*p3;
p1-=3;p3=p2+2;
```

指针移动结果如图 9.7 所示。

$p1$  向地址低的方向移动了 3 个单元,即 12 个字节; $p2$  的指向没有改变; $p3$  向地址高的方向移动了 2 个单元,即 8 个字节。

如果指针的类型是 TYPE,移动  $n$  个单元,则地址实际增加或减少了  $n * \text{sizeof}(\text{TYPE})$  个字节。

② 移动一个单位(前增 1,后增 1,前减 1,后减 1 操作)。

向后(高地址方向)移动一个单元,格式为:

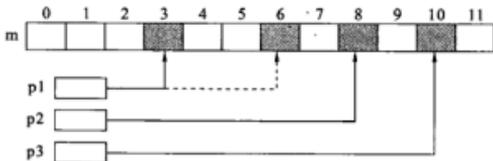


图 9.7 指针移动结果示意图

指针变量++  
或  
++指针变量

向前(低地址方向)移动一个单元,格式为:

指针变量--  
或  
--指针变量

对于前增1和前减1,先改变指针变量的值,使之指向后一个(或前一个)单元,然后再以改变后的指针值作为表达式的值,因此表达式的值和事后变量的值是相同的,例如:

```
int k, *pk=&k;
printf("%0x\n", ++pk); /*显示指针表达式的值*/
printf("%0x\n", pk); /*显示事后指针变量的值*/
```

显示在屏幕上的是两个相同的地址。

格式符%0x,可以用来输出十六进制地址值。

对于后增1和后减1,先以指针变量的值作为表达式的值,然后再改变指针变量的值,使之指向后一个(或前一个)单元,因此表达式的值和事后变量的值是不同的,例如:

```
int k, *pk=&k;
printf("%0x", pk++); /*显示指针表达式的值*/
printf("%0x", pk); /*显示事后指针变量的值*/
```

显示在屏幕上的是两个不同的地址。

**【例 9.2】** 分析下面程序的功能。

```
1 # include <stdio.h>
2 void main()
3 {
4 int *p1, *p2, *p, a, b;
5 scanf("%d, %d", &a, &b);
6 p1=&a; p2=&b;
7 if(a<b)
8 {
```

```

9 p=p1;p1=p2;p2=p;
10 }
11 printf("a=%d,b=%d\n",a,b);
12 printf("max=%d,min=%d\n",*p1,*p2);
13 }

```

程序运行结果:

5,6<回车>

a=5,b=6

max=6,min=5

此例是比较两数的大小,如果 a 小于 b,就通过指针来实现交换数据。注意这里的交换是地址,使指针的指向发生了变化,并没有改变原有变量中的数据,如图 9.8 所示。

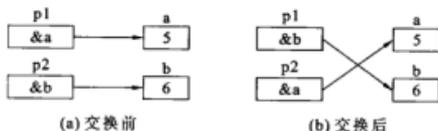


图 9.8 指针交换前后变化示意图

#### 4. 指针变量作函数参数

我们在前面学过,函数可以通过 return 返回一个值,如果要函数返回多个值怎么办?显然用 return 返回语句是办不到的。同时“值传递”过程中实参与形参是彼此独立的存储空间,数据的传递本质上是一种数据的复制,如果形参与实参过多,势必造成内存的额外开销和引起数据复制量的增大,降低了效率。因此,C 语言采用了一种称为指针传递的方式来改变上述两种不足。

通过“传地址”形参指针成为实参指针的副本,于是通过形参指针也可以访问实参指针所指向的数据,因此,指针参数的传递就是把实参指针所指向的数据间接地传递给被调用的函数。

**【例 9.3】** 分析下面的程序。

```

1 #include <stdio.h>
2 void swap(int*p1,int*p2) /*用指针变量实现数据的交换*/
3 {
4 int temp;
5 temp=*p1;
6 *p1=*p2;
7 *p2=temp;
8 }
9 void main()
10 {
11 int a,b,*pointer_1,*pointer_2;

```

```

12 scanf ("%d,%d",&a,&b);
13 pointer_1=&a;pointer_2=&b;
14 if (a<b) //如果 a<b 交换两数
15 swap(pointer_1,pointer_2);
16 printf ("\n%d,%d\n",a,b);
17 }

```

程序运行结果：

```

12,34<回车>
34,12

```

指针变量作为参数，从调用函数向被调用函数传递的不是一个变量，而是变量的地址。同时，从实参向形参的数据传递仍然遵循“单向值传递”的原则，只不过此时传递的是地址，因此对形参的任何操作都相当于对实参的操作。从这个意义上讲指针变量作函数参数的传递又具有了“双向性”，可以带回操作后的结果，如图 9.9 所示。

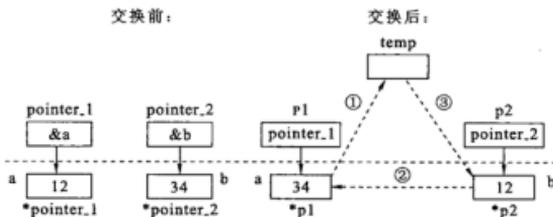


图 9.9 指针作参数交换数据示意图

上面的例子我们也可以改写成下面的形式，请找出不一样的地方。

```

1 # include <stdio.h>
2 void swap (int*p1,int*p2)
3 {
4 int temp;
5 temp=*p1;
6 *p1=*p2;
7 *p2=temp;
8 }
9 void main ()
10 {
11 int a,b;
12 scanf ("%d,%d",&a,&b);
13 if (a<b)
14 swap (&a, &b);
15 printf ("%d,%d\n",a,b);
16 }

```



程序运行结果：

```
5,9<回车>
9,5
```

上述两个程序交换的实质是指针变量所指向的值,即值交换。

**【例 9.4】** 再分析下面的程序并写出结果。

```
1 # include <stdio.h>
2 void swap(int*p1,int*p2) //交换指针变量的地址
3 {
4 int*t; //定义了一个指针变量
5 t=p1;
6 p1=p2;
7 p2=t;
8 printf("%d,%d\n",*p1,*p2); //直接输出 p1,p2 所指向的值
9 }
10 void main()
11 {
12 int a=3,b=5;
13 swap(&a,&b);
14 printf("%d,%d\n",a,b);
15 }
```

程序运行结果：

```
5,3
3,5
```

上例交换的是指针变量的地址,即指针变量的指向交换了,如图 9.10 所示。

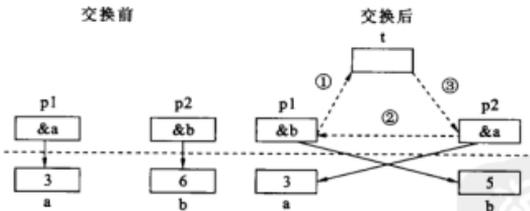


图 9.10 指针变量的指向交换示意图

在上例中,由于 swap 函数中的形参 p1、p2 都是局部变量,p1、p2 的作用域与生存期都被局限在 swap 函数中,不可能将 p1、p2 改变后的值传回到主调函数中,所以主调函数中 a、b 的值不会发生变化。

指针变量作参数,当形参接受了实参所传递的地址后,使形参也指向了实参所指向的单元,形参对其单元的操作,就如同实参对其操作一样,利用这一特点,可以实现用指针变量带回操作后的结果。

**【例 9.5】** 输入长方体的长、宽、高,求长方体体积及正、侧、顶三个面的面积。要求用一个函数实现求体积与面积,并在主函数中实现输入输出操作。

解题思路:这个题前面已经做过,那是用全局变量来实现的。这里我们用指针变量来实现将函数中的值带回来,由于用 return 语句只能带回一个值,所以我们用 return 语句带回体积,用三个指针变量带回三个面的面积。

```
1 #include <stdio.h>
2 int vs(int a,int b,int c,int*s1,int*s2,int*s3)
3 {
4 int v;
5 v=a*b*c;
6 *s1=a*b;
7 *s2=b*c;
8 *s3=a*c;
9 return v;
10}
11
12 void main()
13 {
14 int v,l,w,h;
15 int s1,s2,s3;
16 printf("input length,width and height:");
17 scanf("%d%d%d",&l,&w,&h);
18 v=vs(l,w,h,&s1,&s2,&s3);
19 printf("v=%d\ns1=%d\ns2=%d\ns3=%d\n",v,s1,s2,s3);
20 }
```

程序运行结果:

```
input length,width and height:10 20 30<回车>
v=6000
s1=200
s2=600
s3=300
```

**【例 9.6】** 输入 a、b、c 三个整数,按大小顺序输出。

```
1 #include <stdio.h>
2 void swap(int*a,int*b)
3 {
4 int t;
5 t=*a;
6 *a=*b;
7 *b=t;
```



```
8 }
9
10 void exchange(int*q1,int*q2,int*q3)
11 {
12 if(*q1<*q2)
13 swap(q1,q2);
14 if(*q1<*q3)
15 swap(q1,q3);
16 if(*q2<*q3)
17 swap(q2,q3);
18 }
19
20 void main()
21 {
22 int a,b,c,*p1,*p2,*p3;
23 scanf("%d,%d,%d",&a,&b,&c);
24 p1=&a;
25 p2=&b;
26 p3=&c;
27 exchange(p1,p2,p3);
28 printf("%d,%d,%d\n",a,b,c);
29 }
```

程序运行结果:

9,0,10<回车>

10,9,0

### 9.3.3 数组指针和指向数组的指针变量

一个变量有地址,一个数组也有地址,一个数组还有若干个元素,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和数组元素。

#### 1. 数组指针和数组元素指针

##### 1) 数组指针

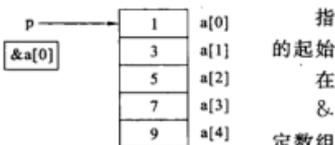
数组指针是指数组的起始地址。

##### 2) 数组元素指针

数组元素指针是指数组元素在内存中的存放地址。



### 3) 指向数组和数组元素的指针变量



指向数组和数组元素的指针变量是指用于存放数组的起始地址或某一数组元素地址的变量。

在图 9.11 中,有: `int a[5], *p;`

`&a[0]` 表示数组第一个元素的地址,同时 C 语言规定数组名代表数组的首地址。因此, `p=a;` 与 `p=&a[0];` 等价,都代表数组的首地址,注意不是代表整个数组。

图 9.11 指向数组的指针示意图

## 2. 一维数组元素的指针访问方式

一维数组的数组名实际上就是指向该数组的第一个单元的指针。一个一维数组若定义为:

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

则数组名 `a` 的类型是 `int*` (数组名代表数组的首地址,因此是指针类型),并且指向第一个元素;因此 `*a` 和 `a[0]` 访问的是同一个元素,两种表达形式完全等价,这种地址表达形式不仅可以访问第一个元素,结合算术运算还可以访问数组的其他元素。例如:

```
* (a+1) 等价于 a[1];
```

```
* (a+2) 等价于 a[2];
```

```
...
```

```
* (a+i) 等价于 a[i].
```

因此,访问数组元素的操作可以采用三种方法:下标法、地址法和指针法。采用指针法比采用地址法更为简洁,执行效率也更高。

指向一维数组第一个元素的指针都可以像一维数组名那样使用,例如:

```
int a[10], *pa=a;
```

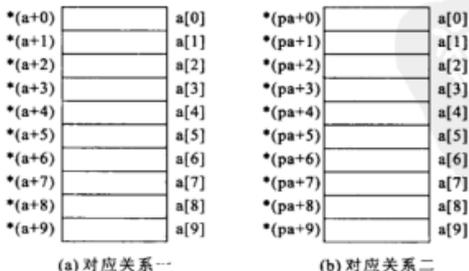
```
* (pa+1) 等价于 a[1];
```

```
* (pa+2) 等价于 a[2];
```

```
...
```

```
* (pa+i) 等价于 a[i].
```

下标法与地址法、下标法与指针法的对应关系如图 9.12 所示。

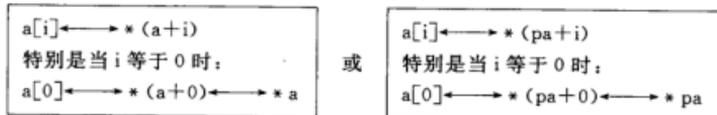


(a) 对应关系一

(b) 对应关系二

图 9.12 数组下标访问法与地址访问法和指针访问法的对应关系

三种访问方式在形式上遵守下面的等价关系：



例如：

```
int a[5]={7,9,4,3,8};
int *pa=a;
printf ("%d ",a[0]);printf ("%d\n",a[2]);
printf ("%d " *a); printf ("%d\n",*(a+ 2));
printf ("%d ",*pa);printf ("%d\n",*(pa+ 2));
```

执行结果：

```
7 4
7 4
7 4
```

指针变量还可以用自增或自减运算来改变指针所指向的位置，达到移动指针的目的。

```
printf ("%d",*(pa++)); printf ("% d",*(++pa));
printf ("%d",*(pa--)); printf ("% d",*(--pa));
```

想一想能否实现：`printf("%d",*(a++));printf("%d",*(a--));`或`printf("%d",*(++a)); printf("%d",*(--a));`；

要特别注意：数组名虽然是指针(地址)，但它是指针常量而不是指针变量，因为我们不可能改变一个常量的值，所以数组名永远指向定义时与之关联的那个数组空间。想用自增或自减来改变数组名的指向是办不到的，即对指针变量的自增自减操作不能用于数组名。

**【例 9.7】** 输出数组的全部元素(几种访问方法比较)。

方法一：下标法。

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

方法二：地址法。

```
main()
{
 int a[10]={1,2,3,4,5};
 int i;
```



```

for(i=0;i<5;i++)
 printf("%d,",*(a+i));
printf("\n");
}

```

方法三:指针法。

```

main()
{
 int a[10]={1,2,3,4,5};
 int*p;
 for(p=a;p<a+5;p++)
 printf("%d,",*p);
 printf("\n");
}

```

三种方法的输出结果都是:

1,2,3,4,5,

**【例 9.8】** 下面的程序的输出结果是什么?

```

1 # include <stdio.h>
2 void main()
3 {
4 int a[10],*p,i;
5 p=a;
6 for(i=0;i<10;i++)
7 scanf("%d",p++); /*特别要注意输入时指针的变化*/
8 printf("\n");
9 for(i=0;i<10;i++,p++) /*指针 p 的值已经变化*/
10 printf("%d ",*p);
11 }

```

程序运行结果:

1 2 3 4 5 6 7 8 9 0<回车>

输出结果不是预期的值!

怎样解决?其实很简单:重新使指针变量指向数组的首地址,请看改进后的例子。

```

1 # include <stdio.h>
2 void main()
3 {
4 int a[10],*p,i;
5 p=a;
6 for(i=0;i<10;i++)
7 scanf("%d",p++);
8 printf("\n");

```



```

9 p=a; /*使 p 指针重新指向了数组的首地址*/
10 for(i=0;i<10;i++,p++)
11 printf("%d ",*p);
12 printf("\n");
13)

```

程序运行结果：

```

1 2 3 4 5 6 7 8 9 0<回车>
1 2 3 4 5 6 7 8 9 0

```

当执行到第 6、7 两行时，随着循环的执行，指针  $p$  的指向在不断的变化，循环结束后，指针  $p$  已指向数组的外面了，如果这时直接用指针来实现输出其值的操作，指针指向的就不是数组的值，而是数组外面的内容，输出结果就不是预期的值。如果在输出之前调整指针  $p$  的指向，使它重新指向数组的首地址，增加第 9 行  $p=a$ ；语句，再输出，就能得到预期的结果。

### 3. 指向数组的指针作函数参数

指向数组的指针也可以作函数参数，它传递的是数组的首地址。这与数组名作函数参数相似。

用数组名和数组指针在参数传递上的组合如图 9.13 所示。

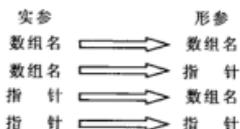


图 9.13 数组名和数组指针作函数参数组合

**【例 9.9】** 将数组  $a$  中的  $n$  个整数按相反的顺序存放。

```

1 # include <stdio.h>
2 void inv(int x[],int n) //数组名作形参
3 {
4 int m,temp,i,j;
5 m=(n-1)/2;
6 for(i=0;i<=m;i++)
7 {
8 j=n-1-i;
9 temp=x[i];x[i]=x[j];x[j]=temp;
10 }
11 }
12
13 void main()
14 {
15 int i,*p,a[10]={3,7,9,11,0,6,7,5,4,2};

```



```

16 p=a;
17 inv(p,10); //数组指针 p 作实参
18 for(i=0;i<10;i++)
19 printf("%d,",a[i]); //下标访问方法
20 printf("\n");
21 }

```

程序运行结果:

```
2 4 5 7 6 0 11 9 7 3
```

还可以将形参修改为指针变量。

```

1 # include <stdio.h>
2 void inv(int*x,int n) //指针变量作形参
3 {
4 int*p,m,t,*i,*j;
5 m=(n- 1)/2;
6 i=x;
7 j=x+n- 1;
8 p=x+m;
9 for(;i<=p;i++,j- -)
10 {
11 t=*i;
12 *i=*j;
13 *j=t;
14 }
15 }
16
17 void main()
18 {
19 int a[10]={3,7,9,11,0,6,7,5,4,2};
20 int*p;
21 p=a;
22 inv(p,10); //数组指针作实参
23 for(p=a;p<a+10;p++)
24 printf("%d,",*p); //指针访问方法
25 printf("\n");
26 }

```

程序运行结果同上。

### 9.3.4 字符串的指针访问法

字符串常量本质上是字符数组,是没有名字(无名)的字符数组。在内存中每个字符

占一个字节,最后以“\0”结束。

由于字符串常量没有名字,因此可以借助于字符数组名和字符指针来表示。

### 1. 用字符数组名表示

```
main ()
{
 char string[]="I love China! ";
 printf("%s\n",string);
}
```

说明:

①string 是数组名,代表数组的首地址;

②string[i]代表数组的第 i 个元素,string[i]等价于 \*(string+i)。其相应的处理方法同一般数组相似。

### 2. 用字符指针表示

```
main ()
{
 char*string="I love China! ";
 printf("%s\n",string);
}
```

说明:

①string 是字符指针,它也代表字符串的首地址。

②下面赋值语句,不是将字符串中的字符赋值给指针变量,而是将其起始地址赋予它。

```
char*str;
str="I am student.";
```

赋值符左边是一个指针变量,右边是一个字符串常量,根据类型匹配原则,右边的字符串常量也应当是一个字符指针(地址)。实际上程序中的字符串常量就是一个指针,是指向存储该字符串的无名字符数组首地址的指针,因此字符串常量“I am student.”的类型是地址。

**【例 9.10】** 将字符串 a 复制到字符串 b。

```
1 # include <stdio.h>
2 void main ()
3 {
4 char a[]="I am boy.",b[20],*p1,*p2;
5 int i;
6 p1=a;
7 p2=b;
8 for(;*p1!='\0';p1++,p2++)
9 *p2=*p1; /*当*p1=='\0'时结束循环,因此'\0'并
 没有复制到*p2上*/
```

```

10 *p2='\0';
11 printf("string a is:%s\n",a);
12 printf("string b is:");
13 for(i=0;b[i]!='\0';i++)
14 printf("%c",b[i]);
15 printf("\n");
16 }

```

程序运行结果:

```

string a is:I am boy.
string b is:I am boy.

```

字符指针和数组名一样也可以作函数参数,既可以作形参也可以作实参。

**【例 9.11】** 用字符数组名作实参。

```

1 void copy_string(char from[],char to[])
2 {
3 int i=0;
4 while(from[i]!='\0')
5 {
6 to[i]=from[i];
7 i++;
8 }
9 to[i]='\0';
10 }
11 # include "stdio.h"
12 void main()
13 {
14 char a[]="I am a teacher.";
15 char b[]="you are a student.";
16 printf("string a=%sstring b=%s\n",a,b);
17 copy_string(a,b);
18 printf("string a=%s\nstring b=%s\n",a,b);
19 }

```

程序运行结果:

```

string a=I am a teacher.string b=you are a student.
string a=I am a teacher.
string b=I am a teacher.

```

**【例 9.12】** 用字符指针作实参。

```

1 void copy_string(char from[],char to[])
2 {
3 int i=0;

```

```
4 while(from[i]!='\0')
5 {
6 to[i]=from[i];
7 i++;
8 }
9 to[i]='\0';
10 }
11 # include "stdio.h"
12 void main()
13 {
14 char a[20]="I am a teacher.";
15 char b[20]="you are a student.";
16 char*p1,*p2;
17 p1=a;
18 p2=b;
19 printf("string a=%sstring b=%s\n",a,b);
20 copy_string(p1,p2);
21 printf("string a=%s\nstring b=%s\n",a,b);
22 }
```

程序运行结果同例 9.11。

**【例 9.13】** 用字符指针作形参。

```
1 void copy_string(char* from,char* to)
2 {
3 for(;*from!='\0';from++,to++)
4 *to=*from;
5 *to='\0';
6 }
7 # include <stdio.h>
8 void main()
9 {
10 char a[20]="I am a teacher.";
11 char b[20]="you are a student.";
12 char*p1=a,*p2=b;
13 printf("string a=%sstring b=%s\n",a,b);
14 copy_string(p1,p2);
15 printf("string a=%s\nstring b=%s\n",a,b);
16 }
```

程序运行结果同例 9.11。



【例 9.14】 设计函数 STRLEN,模拟标准函数 strlen。

```

1 int STRLEN(char*d)
2 {
3 int count=0; /*记录字符串长度*/
4 while(*d++) /*扫描字符串,只要不是结束符,count就增1*/
5 count++;
6 return count;
7 }
8 #include <stdio.h>
9 void main()
10 {
11 printf("%d",STRLEN("abcdefg"));
12 printf("%d",STRLEN("A"));
13 printf("%d\n",STRLEN(""));
14 }

```

程序运行结果:

7,1,0

## 9.4 扩展知识与理论

### 9.4.1 二维数组元素的指针访问方式

上述关于一维数组与指针关系的结论可以推广到二维数组。

前面讲过,二维数组可以看成是一种特殊的一维数组,每一个一维数组元素本身又是一个有若干个数组元素的一维数组。例如:

```
int b[3][4];
```

则这个二维数组可以看成是具有 3 个单元的一维数组,其中每个单元本身又是一个具有 4 个单元的 int 型一维数组。因此,对于二维数组我们仍然可以说“数组名就是指向该数组第一个单元的指针”,只是“第一个单元”的含义不同罢了,如图 9.14 所示:它现在是指构成二维数组第一行的那个一维数组,即  $b[0]$ 。可见数组名  $b$  的类型为  $\text{int} (*)[4]$ ,即一个指向“具有 4 个元素的 int 型一维数组”。这样从二维数组的角度来看,数组名  $b$  代表整个二维数组的首地址,实际上是指二维数组第一行的首地址。因此  $b+1$  代表第二行的首地址, $b+2$  代表第三行的首地址。如果二维数组  $b$  的地址为 3000,则  $b+1$  地址为 3008, $b+2$  的地址为 3016,如图 9.15 所示。

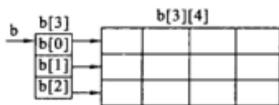


图 9.14 二维数组看成一维数组

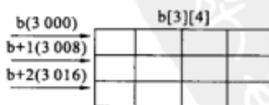


图 9.15 二维数组名与地址的关系

对于二维数组这种特殊的“一维数组”，前面介绍的一维数组下标与指针的等价关系也适用于二维数组。例如，令  $a=b[i]$ ，则有下列变换。

$$b[i][j] \rightarrow a[j] \leftrightarrow *(a+j) \rightarrow *(b[i]+j) \leftrightarrow *((b+i)+j)$$

上述转换的目的是根据二维数组名计算出第  $i$  行第  $j$  列的元素地址。根据这个转换，可以得出一些特殊元素的地址与二维数组名的关系：

第 0 列元素的地址：

$$b[i][0] \leftrightarrow *(b[i]+0) \leftrightarrow *b[i] \leftrightarrow *((b+i)+0) \leftrightarrow *(b+i)$$

第 0 行元素的地址：

$$b[0][j] \leftrightarrow *(b+0)[j] \leftrightarrow (*b)[j] \leftrightarrow *((b+0)+j) \leftrightarrow *(b+j)$$

第 0 行第 0 列元素的地址：

$$b[0][0] \leftrightarrow *(b[0]+0) \leftrightarrow *b[0] \leftrightarrow *(b+0) \leftrightarrow *((b+0)+0) \leftrightarrow *b$$

式子  $*(b+i+j)$  是根据二维数组名计算第  $i$  行第  $j$  列元素的，还有一种直接采用首元素地址计算第  $i$  行第  $j$  列元素的方法。其格式如下：

$$*(\text{首元素地址} + \text{行号} * \text{列数} + \text{列号})$$

二维数组首元素地址是第 0 行第 0 列元素地址。

下面介绍用指向二维数组首行的指针求第  $i$  行第  $j$  列元素的方法。

由于二维数组名表示的是二维数组行地址，因此不能用一般的指向变量的指针来接受二维数组的行地址，要用专门的接受行地址的指针来接受行地址，请注意行指针的定义方法！

```
int b[10][20], (*pb)[20]=b;
```

上述例子中行指针的定义方法，其格式为：

$$\text{类型标识符}(*\text{行指针名})[\text{列}] = [\text{二维数组名}];$$

由于行指针与二维数组名等价，因此很容易得到计算  $i$  行  $j$  列元素的公式，即：

$$*(*(pb+i)+j) \leftrightarrow *((b+i)+j)$$

**【例 9.15】** 设计函数 show\_matrix，它显示参数传来的任意规格的整型二维数组。

解法一：

```
1 #include <stdio.h>
2 void show_matrix(int* array, int row, int col) /* array 为指向变量的
 指针 */
3 {
4 int i, j;
5 for(i=0; i<row; i++)
6 {
7 printf("\n");
8 for(j=0; j<col; j++)
9 printf("%4d", *(array+i*col+j));
10 }
11 }
```

```
12 void main()
13 {
14 int s[4][3]={1,2,3,4,5,6,7,8,9,10,11,12};
15 show_matrix(s[0],4,3);
16 printf("\n");
17 }
```

程序运行结果:

```
1 2 3
4 5 6
7 8 9
10 11 12
```

s[0]是第0行的首地址,当然也是第0行第0列的首地址。

解法二:

```
1 # include <stdio.h>
2 void show_matrix(int*array,int row,int col)
 /* array为指向变量的指针*/
3 {
4 int i,j;
5 for(i=0;i<row;i++)
6 {
7 printf("\n");
8 for(j=0;j<col;j++)
9 printf("%4d",*(array+i*col+j));
10 }
11 }
12 void main()
13 {
14 int s[4][3]={1,2,3,4,5,6,7,8,9,10,11,12};
15 show_matrix(&s[0][0],4,3);
16 printf("\n");
17 }
```

程序运行结果同解法一。

如果用首行首列元素地址作实参,一定不要忘记使用取地址符“&”。

解法三:

```
1 # include <stdio.h>
2 void show_matrix(int (*array)[3],int row,int col)
 /*array为行指针*/
3 {
4 int i,j;
```

```

5 for(i=0;i<row;i++)
6 {
7 printf("\n");
8 for(j=0;j<col;j++)
9 printf("%4d",*(*(array+i)+j));
 /*计算元素公式变了*/
10 }
11 }
12 void main()
13 {
14 int s[4][3]={1,2,3,4,5,6,7,8,9,10,11,12};
15 show_matrix(s,4,3); //二维数组名作实参
16 printf("\n");
17 }

```

程序运行结果同解法一。

array 为行指针,用来接受首行地址,注意计算第 i 行第 j 列元素的公式也变了。

## 9.4.2 指针数组与带参数的 main 函数

前面学过,凡是具有数组首地址的指针都可以称为指向数组的指针或称为数组指针。什么是指针数组呢?

193

### 1. 指针数组

若数组的每一个元素都是一个指针,这样的数组称为指针数组。

定义一维指针数组格式为:

|                   |
|-------------------|
| 类型标识符 *数组名[数组长度]; |
|-------------------|

或

|                           |
|---------------------------|
| 类型标识符 *数组名[数组长度][={初值表}]; |
|---------------------------|

定义二维指针数组格式为:

|                     |
|---------------------|
| 类型标识符 *数组名[行数][列数]; |
|---------------------|

或

|                           |
|---------------------------|
| 类型标识符 *数组名[行数][列数]={初值表}; |
|---------------------------|

与一般的数组的定义相比,指针数组只是在数组名前增加了一个 \* 号,这里的初值表是一系列用逗号隔开的指针表达式。例如:

```

int *ip[4];float*fp[3];
double*dp[2][3];

```

```
char *weekday[7]={"Sunday","Monday","Tuesday","Wednesday",
 "Thursday","Friday","Saturday"};
```

也可以用二维数组来表示上面指针数组 weekday,其定义方法为:

```
char week[7][10]={"Sunday","Monday","Tuesday","Wednesday",
 "Thursday","Friday","Saturday"};
```

它在内存中的存储结构如图 9.16 所示。

|   |   |   |   |   |   |    |    |    |    |
|---|---|---|---|---|---|----|----|----|----|
| S | u | n | d | a | y | \0 |    |    |    |
| M | o | n | d | a | y | \0 |    |    |    |
| T | u | e | s | d | a | y  | \0 |    |    |
| W | e | d | n | e | s | d  | a  | y  | \0 |
| T | h | u | r | s | d | a  | y  | \0 |    |
| F | r | i | d | a | y | \0 |    |    |    |
| S | a | t | u | r | d | a  | y  | \0 |    |

图 9.16 二维数组占用内存空间

该数组一共占用了 70 个字节。从上面的例子可以看出,如果采用二维数组来定义将会造成一定的存储空间浪费(上述例子浪费 13 个字节)。

如果用指针数组来表示,由于指针数组的每个元素都是指针,因此它们可以指向字符串的首地址,通过这个首地址可以访问该字符串,而且相对二维数组可以节省内存空间,如图 9.17 所示。



图 9.17 指针数组占用内存空间

**注意:**(1) 字符串并不是一个二维数组,字符串常量的存储空间不一定是连续的(不要将其理解为也是二维数组),这一点与二维数组存放字符串不一样,而指针数组的存储空间是连续的。

(2) 指针数组中的每一个数组元素指向一个字符串的首地址,因此可以通过它来访问字符串。

可以从下面例子中看到用指针数组来显示字符串。

**【例 9.16】** 用指针数组显示字符串常量。

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

```
3 {
4 int i;
5 char *weekday[7]={"Sunday","Monday","Tuesday","Wednesday",
 "Thursday","Friday","Saturday"};
6 for(i=0;i<7;i++)
7 printf("%s ",weekday[i]);
8 }
```

程序运行结果:

Sunday Monday Tuesday Wednesday Thursday Friday Saturday

另外,还可以利用指针数组来为字符串排序,这样可以不必移动字符串的位置,只需改动指针数组中各元素的指向,而且改变指针变量的值(地址)比移动字符串所花的时间少得多。

**【例 9.17】** 将若干个字符串按字母顺序(由小到大)输出。

```
1 # include <stdio.h>
2 # include <string.h>
3 void main()
4 {
5 void sort(char*name[],int n);
6 void print(char*name[],int n);
7 char*name[]={"Follow me","BASIC","Great Wall","FORTRAN",
 "Computer design"};
8 int n=5;
9 sort(name,n);
10 print(name,n);
11 }
12 void print(char*name[],int n)
13 {
14 int i;
15 for(i=0;i<n;i++)
16 printf("%s\n",name[i]);
17 }
18 void sort(char*name[],int n)
19 {
20 char*t;
21 int i,j,k;
22 for(i=0;i<n-1;i++)
23 {
24 k=i;
25 for(j=i+1;j<n;j++)
26 if(strcmp(name[k],name[j])> 0)
```



```

27 k=j;
28 if (k!=i)
29 {
30 t=name[i];
31 name[i]=name[k];
32 name[k]=t;
33 }
34 }
35 }

```

程序运行结果:

```

BASIC
Computer design
FORTRAN
Follow me
Great Wall

```

程序运行前后指针指向如图 9.18、图 9.19 所示。

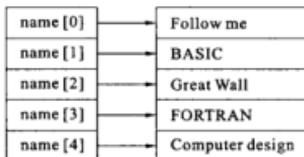


图 9.18 程序运行前指针指向

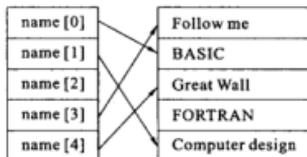


图 9.19 程序运行后指针指向

## 2. 带参数的 main 函数和 void 指针

### 1) 带参数的 main 函数

除主函数外,函数是可以互相调用的,主函数可以调用其他函数,其他函数不能调用主函数,主函数是被系统调用的。函数调用时可以带参数也可以不带参数,这完全取决于该函数是有参函数还是无参函数。主函数可不可以带参数呢?回答是肯定的,这种参数是在程序运行时给出的,所以又称为命令行参数。下面来讨论带参数的主函数。

带参数的 main 函数原型如下:

```
[返回类型] main(int argc,char * argc[]);
```

**返回类型:**返回值提供给运行该程序的操作系统,如果返回值为 void,表示不需要返回值,如果缺省返回值类型,表示返回值类型为整型。

主函数有下面两个参数。

- (1) argc:为整型数据变量,表示命令行中提供的实参个数(包括命令本身)。
- (2) argv:为指针数组变量,该数组的大小由命令行参数的个数决定,该数组是指向

命令行字符串的指针数组。

特别要注意的是：

argv[0]指向程序(命令)名；

argv[1]指向命令行第一个参数；

...

argv[argc-1]指向命令行最后一个参数。

程序运行格式为：

|                             |
|-----------------------------|
| 程序名    [参数 1 参数 2 ... 参数 n] |
|-----------------------------|

**【例 9.18】** 显示命令行参数的数量和每个命令行参数(源文件名用 file1.c)。

```

1 # include <stdio.h>
2 main(int argc, char* argv[])
3 {
4 int i;
5 printf("有%d个命令行参数:\n",argc);
6 for(i=0;i<argc;i++)
7 printf("argv[%d]:%s\n",i,argv[i]);
8 }

```

程序运行结果：

D:\> file1 China Beijing<回车>

输出:有 3 个命令行参数:

```

argv[0]:file1
argv[1]:China
argv[2]:Beijing

```

如果参数本身含有空格,须用双引号括起来。例如,

输入:D:\file1 "China Beijing" "China Shanghai"

输出:有 3 个命令行参数。

```

argv[0]:file1
argv[1]:China Beijing
argv[2]:China Shanghai

```

## 2) void 指针

void 指针为空类型指针,它的定义格式为:

|                  |
|------------------|
| void    * 指针变量名; |
|------------------|

空类型指针指向的不是一个有确定类型的数据,它和一般的指针变量有以下不同之处。

(1) 不能直接引用 void 指针所指向的数据,因为数据类型没有确定,不能执行任何操作,必须先进行相应的类型转换后才能引用,这种转换是强制类型转换。例如:

```
void *p;
```

```
p=30; / 错误的使用方法 */
(int)p=30; /* 正确的使用方法 */
```

(2) 在进行强制类型转换后 void 指针变量和其他类型的指针变量允许进行赋值。

例如：

```
void *p;
int *q;
p=(void*)q;
q=(int*)p;
```

void 指针最大的好处是可以作为通用类型的指针变量，允许保存任何类型的指针，但要牢记，在使用数据前一定要进行适当的类型转换。

**【例 9.19】** void 指针的应用。

```
1 #include <stdio.h>
2 void swap(void*x,void*y)
3 {
4 int temp;
5 temp=*(int*)x;
6 *(int*)x=*(int*)y;
7 *(int*)y=temp;
8 }
9 void main()
10 {
11 int a=5,b=3;
12 printf("a=%d,b=%d\n",a,b);
13 swap(&a,&b);
14 printf("a=%d,b=%d\n",a,b);
15 }
```

程序运行结果：

```
a=5,b=3
a=3,b=5
```

### 9.4.3 常见错误及处理方法

常见错误 1: 指针变量类型与它所指向的数据类型不一致。

C 语言规定，指针变量的类型是它所指向的数据类型，如果指针变量类型与它所指向的数据类型不一致，系统将给出警告性错误，运行结果也将出错。

常见错误 2: 用一般数据给指针变量赋值。

例如：`int * p=2000`；此时计算机将 2000 当作数值而非地址，因此系统将给出警告性错误。正确赋值方法应当是下列三种方法之一。

用变量地址：`int x, * p=&x`；

用相同类型的指针赋值：`int x, * p=&x; int * q;q=p`；

用数组名或字符串常量赋值: `int a[5], *p;p=a;char *p="china";`

常见错误 3:不能正确地表示指针所指向的值。

要得到指针所指向的值,就必须进行指针间接运算,如果指针变量用 `p` 来表示,`p` 所指向的值就表示成 `*p`。

常见错误 4:不能正确理解 `*p++`、`(*p)++`、`*++p`、`++*p` 表达式的值与指针指向的变化。

`*p++`,由于 `*` 与 `++` 运算符的优先级相同,按右结合性原则,首先进行间接运算 `*p`,取 `p` 所指向单元的数值作为表达式的值,再进行 `p++` 运算,使 `p` 指向下一个单元。

`(*p)++`,由于括号运算符的优先级较高,首先进行间接运算 `*p`,取 `p` 所指向单元的数值作为表达式的值,然后使该单元的值增 1。指针变量的指向不变。

`*++p`,由于两运算符的优先级相同,按右结合性原则,先进行指针移动运算 `++p`,使 `p` 指向下一个单元,然后再取该单元的值并将该值作为表达式的值。

`++*p`,由于两运算符的优先级相同,按右结合性原则,先进行间接运算 `*p`,然后将指针变量所指向的单元值加 1,并将该值作为表达式的值,指针变量的指向不变。

常见错误 5:不能正确区分下列两种初始化语句。

```
char str[]={'c','h','i','n','a','\0'}; //正确
char*pstr={'c','h','i','n','a','\0'}; //错误
```

定义字符数组后,系统就在内存中开辟相应的存储空间用来存放初始字符,数组名代表这个存储空间的首地址。定义指针变量后,系统只开辟存放一个字符地址的存储空间,这个地址空间不可能存放初始化的字符,因此第一条初始化语句是正确的,第二条语句是错误的。如果用字符串常量初始化上述两条语句,它们又是正确的,请注意区别。

```
char str[]="china"; //正确
char*pstr="china"; //正确
```

常见错误 6:指针变量在使用前没有指向一个正确的存储单元。

指针变量与其他变量一样,如果只定义不初始化,系统将给一个随机值作为变量的初值,指针变量的随机值是一个随机的地址,这个随机地址有可能是系统重要数据的地址,如果用一些语句改变了这些重要数据,将有可能造成系统的崩溃,这样就会带来灾难性后果。避免的方法就是在使用指针变量之前初始化该指针变量,如果暂时不能初始化,也可以给指针变量赋 `NULL`。

常见错误 7:直接给字符指针输入字符串。

```
char str[81];
scanf("%s",str); //正确
char*pstr;
scanf("%s",pstr); //错误
```

利用数组名和 `%s` 格式符可以直接给字符数组输入字符串,因为,定义数组时系统就开辟了相应的存储空间,它可以以从数组名表示的首地址开始依次存放输入的字符。而直接用字符指针变量名和 `%s` 格式符就不能完成输入操作,这是因为,字符指针变量只有首地址,而没有存储空间,因此不可能存储输入的字符,修改的办法就是要使字符指针变量指向一个连续的存储空间。

```

char str[81]; //定义一个连续的存储空间
char *pstr;
pstr=str; //使指针变量指向这个连续的存储空间
scanf("%s",pstr);

```

## 9.5 深入训练

- (1) 用指针来实现求三个整数的最大值和最小值。
- (2) 用指针来实现两个整数的交换。
- (3) 编程用指针实现求字符串长度(StrLen())、求两字符串相连(StrCat())、求两字符串复制(StrCpy())、字符串比较(StrCmp())等函数的功能。
- (4) 编程实现用指针变量在一个已知的成绩数组中,查找给定成绩并输出这些成绩的函数,要求输入、查找和输出操作作用不同的函数实现。
- (5) 编程实现定义、输入、输出五个学生姓名的函数。要求用指针实现输入/输出操作,并在不同的函数中实现。
- (6) 编程实现用指针变量在函数之间传递字符串的函数。
- (7) 用指针变量编程实现查找三个字符串中最大的字符串操作。
- (8) 用指向成绩数组的指针变量做函数参数,实现求学生成绩平均分和总分数的操作。
- (9) 用指针变量编程实现输入输出多个学生姓名的操作。
- (10) 有 10 个学生成绩组成的数组,用带指针变量参数的函数,计算该数组学生成绩总分和平均分,然后用指针变量带回总分和平均分,并在主函数中输出总分和平均分。

## 习 题 9

### 9.1 填空题

- (1) C 语言的取地址符是\_\_\_\_\_。
- (2) 定义指针变量时必须在变量名前加\_\_\_\_\_,指针变量是存放\_\_\_\_\_变量。
- (3) 内存地址是从\_\_\_\_\_开始编号的,因此\_\_\_\_\_是最小的指针值。
- (4) 指针变量的类型是指\_\_\_\_\_。
- (5) 已知整型变量 k 定义为 int k;,指向变量 k 的指针变量定义方法是\_\_\_\_\_。
- (6) p 是一个指针变量,取 p 所指向单元的数据作为表达式的值,然后使 p 指向下一个单元的表达式是\_\_\_\_\_。
- (7) 设指针 p 定义为:int array[ ]={5,10,15};\*p=array;
  - \* p++的运算后,表达式的值是\_\_\_\_\_,指针 p 指向\_\_\_\_\_。
  - (\* p)++运算后,表达式的值是\_\_\_\_\_,指针 p 指向\_\_\_\_\_。
  - \* ++p 的运算后,表达式的值是\_\_\_\_\_,指针 p 指向\_\_\_\_\_。
  - ++ \* p 的运算后,表达式的值是\_\_\_\_\_,指针 p 指向\_\_\_\_\_。

- (8) 数组名代表数组的\_\_\_\_\_,但数组名是\_\_\_\_\_常量。
- (9) 一维数组的下标访问方式是\_\_\_\_\_,对应的指针访问方式是\_\_\_\_\_。
- (10) 已知一维数组 `float array[5];`,指向一维数组的指针变量定义方法是\_\_\_\_\_。
- (11) 已知二维数组 `double X[4][7];`,指向二维数组的指针变量定义方法是\_\_\_\_\_。
- (12) 定义一维数组是在定义数组时在数组名前加\_\_\_\_\_。

(13) 根据图 9.20 填空。

建立如图所示存储结构所需的说明语句是\_\_\_\_\_。

建立如图所示给 a 输入数据的输入语句是\_\_\_\_\_。

建立如图所示存储结构所需的赋值语句是\_\_\_\_\_。



图 9.20

### 9.2 判断题(判断下列叙述的正确性,正确的请打“√”,错误的请打“×”)

- (1) 内存单元地址实际是内存单元的编号。 ( )
- (2) 指针也是地址,它是特定数据的地址。 ( )
- (3) 指针变量是变量,它可以存放地址,也可以存放数据。 ( )
- (4) a 是一维数组名, \*a+1 与 a[1] 等价。 ( )
- (5) 只有将变量的地址送给了指针变量,指针变量才会指向该变量。 ( )
- (6) b 是二维数组名, b+1 是第二个(b[0][1])元素的地址。 ( )

### 9.3 选择题

(1) 下列说法不正确的是( )。

- A. 一个数组的地址就是该数组的第一个元素(0 元素)的地址
- B. 地址 0 专用于表示空指针
- C. 地址值 0 可以用符号常量 NULL 表示
- D. 两个指针相同是指它们的地址值相同

(2) 空指针是指( )。

- A. 无具体指针值的指针
- B. 不指向任何数据的指针
- C. 无数据类型的指针
- D. 既无指针值又无数据类型的指针

(3) 有如下语句 `int a=10,b=20,*p1,*p2;p1=&a;p2=&b;`如图 9.21 所示;若要实现图 9.22 所示的存储结构,可选用的赋值语句是( )。

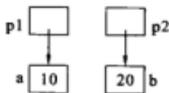


图 9.21

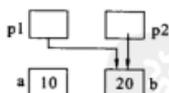


图 9.22

- A. \*p1 = \*p2;    B. p1 = p2;    C. p1 = \*p2;    D. \*p1 = p2;

(4) 下面程序段中,打印出星号的数目是( )。

```
char*s="\ta\018bc";
for(;*s!='\0';s++)
printf("**");
```

- A. 9                      B. 7                      C. 6                      D. 5

(5) 若定义: `int a[3][4]`; 下列选项不能表示数组元素 `a[1][1]` 地址的是( )。

- A. `a[1]+1`      B. `&a[1][1]`      C. `*(a+1)[1]`      D. `*(a+5)`

(6) 以下程序的运行结果是( )。

```
#include <stdio.h>
sub(int x,int y,int *z)
{
 *z=y-x;
}
main()
{
 int a,b,c;
 sub(10,5,&a);
 sub(7,a,&b);
 sub(a,b,&c);
 printf("%4d,%4d,%4d\n",a,b,c);
}
```

- A. 5, 2, 3      B. -5, -12, -7  
C. -5, -12, -17      D. 5, -2, -7

(7) 下列程序段的运行结果是( )。

```
char *p="%d,a=%d,b=%d\n";
int a=111,b=10,c;
c=%b;
p+=3;
printf(p,c,a,b);
```

- A. 1,a=111,b=10    B. a=1,b=111    C. a=111,b=10    D. 以上结果都不对

(8) 下面程序段的运行结果是( )。

```
char *s="abcde";
s+=2;
printf("%s",s);
```

- A. cde      B. 字符 c      C. 字符 c 的地址      D. 无确定的输出结果

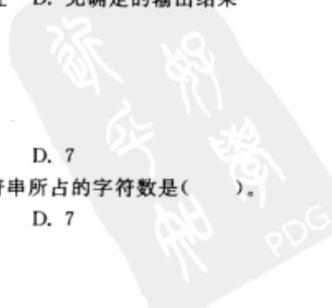
(9) 下面程序段中,for 循环的执行次数是( )。

```
char *s="\ta\018bc";
for(;*s!='\0';s++)
 printf("**");
```

- A. 9      B. 5      C. 6      D. 7

(10) 设 `char *s="\ta\017bc"`; 则指针变量 `s` 指向的字符串所占的字符数是( )。

- A. 9      B. 5      C. 6      D. 7



## 单元 10 项目中结构体的应用

### 能力目标

- 能将学生信息定义成结构体类型,能根据该类型定义结构体变量并进行初始化。
- 能计算结构体变量占用空间大小,利用“.”运算符和“->”运算符正确引用结构体成员变量。
- 能用结构体变量作函数参数在函数之间传递结构体数据。
- 能用结构体类型定义结构体数组并初始化,能用下标访问法访问结构体数组元素中的成员变量,能将结构体数组名设计成函数参数并进行数据传递。
- 能正确定义和访问“班级学生成绩管理系统”中学生信息结构体数组。

### 知识目标

- 理解产生结构体类型的数据的原因。
- 掌握结构体类型和结构体变量定义的三种方法,理解结构体变量和结构体成员变量的区别与联系,掌握结构体成员变量的引用方法。
- 理解结构体数组与一般数组的相同与不同点,理解数组的下标访问法和指针访问法在结构体数组中的应用。
- 理解结构体数组名和指向结构体数组的指针变量作函数参数的相同点与不同点。

### 学习提示

结构体是程序设计中应用最为广泛的一种数据类型,它主要应用于不同类型的若干数据整体存储与访问。由于结构体类型是用户自定义类型,所以,学习结构体与前面学习简单类型有些不一样,结构体要先定义类型,再定义变量、数组等,然后才能访问其成员,访问成员方法又分为变量访问方法和指针访问方法。

## 10.1 任务 14:用结构体实现数据的增加、删除、修改和显示

结构体类型是“班级学生成绩管理系统”中学生属性的主要类型,学生属性就是用结

构体类型来实现的。本任务主要介绍增加、删除、修改、显示全部学生记录函数。

- (1) 首先定义学生数组长度和函数的声明,假定要处理的学生不超过40人。

```
define STUSIZE 40
int Add(struct student*,int*); //追加记录函数
int Del(struct student*,int*); //删除记录函数
int Modify(struct student*,int*); //修改记录函数
void DispAll(struct student*,int,char*); //显示全部记录函数
```

- (2) 再定义学生信息结构体类型。

```
struct student
{
 int stunum; //学号
 char stuname[10]; //学生姓名
 float stuscore[5]; //3门成绩、平均成绩、总成绩
};
```

(3) 在主函数中第一个清屏之前,定义能存储40个学生信息的数组和记录当前学生数的整型变量stunum。

```
struct student stu[STUSIZE]; //定义学生数组
int stunum=0; //用来记录当前学生记录数
```

- (4) 在主函数中将增加、删除、修改、显示函数调用修改为:

```
Add(stu,&stunum);
Del(stu,&stunum);
Modify(stu,&stunum);
DispAll(stu,stunum,"显示全部学生记录");
```

- (5) 分别将增加、删除、修改、显示函数重新定义为:

```
int Add(struct student stu[],int* size) //增加学生记录函数
{
 int i,j;
 int stunum;
 int number;
 system("cls");
 if(*size>=40) //判断数组是否已经装满
 {
 gotoxy(30,2);
 printf("数组已满,不能再增加记录!");
 return 0;
 }
 else
 {
 do //判断输入的增加记录数是否合适
```

```
{
 gotoxy(30,2);
 printf("请输入增加的记录个数:");
 scanf("%d",&number);
 if(number<0 || number+*size>=40)
 {
 gotoxy(30,2);
 printf("输入增加记录个数错,请重新输入!");
 }
}while(number<0 || number+*size>=40);
stunum=*size+number;
system("cls");
gotoxy(33,2);
printf("学生信息输入!");
for(i=*size;i<stunum;i++) //增加学生记录
{
 gotoxy(15,5);
 printf("请输入第%d个学生学号:",i+1);
 gotoxy(15,7);
 printf("请输入第%d个学生姓名:",i+1);
 gotoxy(15,9);
 printf("请输入第1门成绩:");
 gotoxy(15,11);
 printf("请输入第2门成绩:");
 gotoxy(15,13);
 printf("请输入第3门成绩:");
 gotoxy(37,5);
 scanf("%d",&stu[i].stunum);
 gotoxy(37,7);
 scanf("%s",stu[i].stuname);
 for(j=0;j<3;j++)
 {
 gotoxy(32,9+j*2);
 scanf("%f",&stu[i].stuscore[j]);
 }
}
if(i==*size)
{
 gotoxy(33,4);
```



```

 printf("没有记录输入!");
 }
 *size=stunum; //用指针变量带回学生记录数
 gotoxy(33,15);
 printf("按任意键返回上级菜单!");
 getch();
 return 1;
}
}

```

该函数有一个判断数组是否装满的语句,只有在数组没有装满的情况下才能增加学生记录。由于执行增加学生记录操作后,学生数会发生变化,因此,用一个指针变量作参数来返回变化了的的学生记录。

```

int Del(struct student stu[],int*stusize) //删除学生记录函数
{
 int i,k;
 int number;
 int loop=0; //学号输入正确标志
 system("cls");
 gotoxy(33,2);
 printf("删除学生记录!\n");
 if(*stusize<=0)
 {
 gotoxy(20,4);
 printf("数组中没有学生记录或文件没有打开,不能删除记录!");
 getch();
 return 0;
 }
 else
 {
 do //找出删除学生记录的下标
 {
 system("cls");
 gotoxy(25,2);
 printf("删除学生记录(不删除记录请输入-1)!\n");
 gotoxy(28,4);
 printf("请输入被删除学生的学号:");
 scanf("%d",&number);
 if(number==-- 1)
 {

```

```

 return 0;
 }
 for(i=0,k=0;i<*stusize;i++)
 {
 if(number==stu[i].stunum)
 {
 loop=1;
 k=i; //被删除记录的下标
 break;
 }
 }
 if(loop!=1)
 {
 gotoxy(25,6);
 printf("输入学生学号出错,按任意键重新输入!");
 getch();
 }
 }while(loop!=1);
}
for(i=k;i<*stusize;i++) //删除操作
{
 stu[i]=stu[i+1];
}
gotoxy(25,6);
printf("删除成功,按任意键返回上级菜单!");
*stusize=*stusize-1;
getch();
return 1;
}

```

如果没有打开文件或文件中没有学生记录就不能删除学生记录。如果输入了-1,就表示不删除记录,如果输入学号出错就给出提示。删除学生记录的方法,是从被删除记录开始,用后一个记录覆盖前一个记录,直到记录结束。

```

int Modify(struct student stu[],int*stusize) //修改学生记录函数
{
 int i,k;
 int number;
 int loop=0; //学号输入正确标志
 system("cls");
 gotoxy(33,2);

```

```
printf("修改学生记录!\n");
if(*stusize<=0)
{
 gotoxy(20,4);
 printf("数组中没有学生记录或文件没有打开,不能修改记录!");
 getch();
 return 0;
}
else
{
 do //找出修改学生记录的下标
 {
 system("cls");
 gotoxy(25,2);
 printf("修改学生记录!(不修改请输入-1表示)\n");
 gotoxy(28,4);
 printf("请输入被修改学生的学号:");
 scanf("%d",&number);
 if(number== -1)
 {
 return 0;
 }
 for(i=0,k=0;i<*stusize;i++)
 {
 if(number==stu[i].stunum)
 {
 loop=1;
 k=i; //被修改记录的下标
 break;
 }
 }
 if(loop!=1)
 {
 gotoxy(25,6);
 printf("输入学生学号出错,按任意键重新输入!");
 getch();
 }
 }while(loop!=1);
}
```



```
system("cls");
gotoxy(33,2);
printf("修改学生记录!\n");
gotoxy(28,4);
printf("学号:%d",stu[k].stunum);
gotoxy(28,6);
printf("姓名:%s",stu[k].stuname);
gotoxy(28,8);
printf("成绩 1:%.1f",stu[k].stuscore[0]);
gotoxy(28,10);
printf("成绩 2:%.1f",stu[k].stuscore[1]);
gotoxy(28,12);
printf("成绩 3:%.1f",stu[k].stuscore[2]);
gotoxy(34,4);
scanf("%d",&stu[k].stunum);
gotoxy(34,6);
scanf("%s",&stu[k].stuname);
gotoxy(35,8);
scanf("%f",&stu[k].stuscore[0]);
gotoxy(35,10);
scanf("%f",&stu[k].stuscore[1]);
gotoxy(35,12);
scanf("%f",&stu[k].stuscore[2]);
gotoxy(25,14);
printf("修改成功,按任意键返回上级菜单!");
getch();
return 1;
}
```

```
void DispAll(struct student stu[],int size,char str[])
```

//显示全部记录函数

```
{
 int i,j;
 system("cls");
 if(size<=0)
 {
 gotoxy(20,4);
 printf("数组中没有学生记录或文件没有打开,不能显示记录!");
 getch();
 }
}
```



```
else
{
 gotoxy(30,2);
 printf("%s\n",str);
 gotoxy(5,4);
 printf("学号 姓名 成绩1 成绩2 成绩3 总成绩
 平均成绩");
 for(i=0;i<size;i++)
 {
 gotoxy(5,6+i);
 printf("%-5d",stu[i].stunum);
 printf("%8s",stu[i].stuname);
 for(j=0;j<5;j++)
 {
 printf("%10.1f",stu[i].stuscore[j]);
 }
 printf("\n");
 }
 gotoxy(28,7+size);
 printf("按任意键返回上级菜单!");
 getch();
}
}
```

单元能力训练任务分别如下。

任务 A:按“班级学生成绩管理系统”项目要求,将学生信息定义成结构体类型和结构体变量。

任务 B:用学生信息结构体类型定义含有 40 个学生的结构体数组。

任务 C:用学生信息结构体数组名或指向该数组名的指针变量作函数参数,实现学生记录的增加、删除、修改、显示操作。

任务 D:用学生信息结构体数组名或指向该数组名的指针变量作函数参数,实现学生信息的查找、排序、计算等操作。

任务 E:模仿任务 14 实现“学生通讯录”的学生信息结构体类型及其变量、数组的定义,以及相关的操作。

## 10.2 必备知识与理论

### 10.2.1 结构体概述

前面学习了一些基本类型(也称为简单类型),如整型、实型、字符型等,这些类型都是



系统定义好的,程序员可以直接拿来定义变量。世界是复杂的,事务的内在关系也是复杂的。例如,一个学生的属性包括学号、姓名、性别、出生日期、学习成绩、家庭住址等,这些数据虽然可以用基本类型来定义,但是这样定义不是一个有机的整体,不符合客观实际。因此,只用基本类型来表述这些事务显然是不够的,也就是说基本类型不能全面反映客观世界。

既然系统没有定义这些复杂的类型,那么用户能不能根据客观实际来定义数据类型呢?C语言提供了自定义数据类型的方法,通过自定义类型将不同类型的数据组合成一个有机的整体,以便访问。这些数据在这个整体中是互相联系的,这种自定义的数据类型称为构造类型。实际上在前面已经学习了一种构造类型——数组,数组是具有相同数据类型的一组元素集合。除了数组之外还有结构体和共用体,它们主要用来处理不同类型若干数据的整体存储与访问。

## 10.2.2 结构体类型的应用

### 1. 结构体类型的定义

结构体类型就是将不同类型的数据组合成一个有机的整体,以便于访问。如图 10.1 所示。

| num   | name   | sex | age | score | addr    |
|-------|--------|-----|-----|-------|---------|
| 10010 | Li Fun | M   | 18  | 87.5  | Beijing |

(将不同类型的数据组合成一个有机的整体)

图 10.1 设计学生属性

设一个学生的属性:学号(num)、姓名(name)、性别(sex)、年龄(age)、成绩(score)、家庭住址(addr)。

很显然,C语言没有提供这种现成的数据类型,因此用户必须要在程序中自己建立所需的结构体类型,这种类型就是用户自定义的结构体类型。

定义一个结构体类型的一般格式为:

```
struct 结构体类型名
{
 成员列表
};
```

其中,struct 是定义结构体类型的关键字,结构体类型名由用户自己设计,但要遵循 C 语言标识符的命名原则,成员列表可以是简单类型变量、指针变量、数组和除自身类型之外的已定义的结构体类型变量。

以上述学生属性为例,来建立一个结构体类型。

```
struct student
{
 int num;
 char name[20];
```

```

char sex;
int age;
float score;
char addr[30];
};

```

上面定义了一个称为 struct student 的结构体类型,它包括 num、name、sex、age、score、addr 等不同类型的数项,定义结构体类型时一定要注意下面几个问题。

(1) 结构体类型名是 struct student,而不是 student。它和系统提供的基本类型一样具有同样的地位和作用,都是可以用来定义变量的类型。

(2) 在大括号中定义的变量称为成员,其定义方法和前面变量定义的方法一样,只是我们不能忽略右大括号后面的分号。

## 2. 结构体变量的定义与初始化

### 1) 结构体变量的定义

定义好结构体类型之后,就可以定义结构体变量了。定义结构体变量有三种方法。

(1) 先定义结构体类型再定义结构体变量。

格式为:

```

struct 结构体类型名
{
 成员列表
};
struct 结构体类型名 变量名列表;

```

例如:

```

struct student
{
 int num;
 char name[20];
 char sex;
 int age;
 float score;
 char addr[30];
};
struct student stu1,stu2;

```

结构体类型只是一个模型,并无具体的数据,系统也不对它分配内存单元,系统只对变量分配内存空间。即系统对 stu1 和 stu2 分配内存空间。

下面的问题是,系统为 stu1 和 stu2 分配多大的空间?

结构体变量所占存储空间的大小,是成员列表中所有成员所占内存空间之和。例如:

|       |       |           |   |    |      |          |
|-------|-------|-----------|---|----|------|----------|
| stu1: | 10001 | Zhang Xin | M | 19 | 90.5 | Shanghai |
| stu2: | 10002 | Wang Li   | F | 20 | 98   | Beijing  |

该结构体所有成员占用字节数=4+20+1+4+4+30=63,即 stu1 和 stu2 各占内存空间 63 个字节。

(2) 定义结构体类型的同时定义结构体变量。

格式为:

```
struct 结构体类型名
{
 成员列表;
}变量名列表;
```

例如:

```
struct student
{
 int num;
 char name[20];
 char sex;
 int age;
 float score;
 char addr[30];
}stu1,stu2;
```

定义后变量在内存中开辟的空间与第 1 种定义方法相同。

(3) 直接定义结构体类型变量(匿名定义)。

格式为:

```
struct
{
 成员列表;
}变量名列表
```

其特点是在定义时不出现结构体类型名。

例如:

```
struct
{
 int num;
 char name[20];
 char sex;
 int age;
 float score;
```



```

 char addr[30];
 }stu1,stu2;

```

定义后变量在内存中开辟的空间与第 1 种定义方法相同。

关于结构体类型说明如下：

- ① 类型与变量是两个不同的概念，不能混淆。变量分配内存空间，类型不分配空间；
- ② 结构体成员可以单独使用，相当于普通变量，访问方法后面具体讲述；
- ③ 结构体成员也可以是一个已定义好类型的结构体变量。如将上述结构体年龄成员改为生日成员。

成员改为生日成员。

```

struct date //日期结构体
{
 int year;
 int month;
 int day;
};
struct student1
{
 int num;
 char name[20];
 char sex;
 float score;
 struct date birthday; /*birthday是struct date类型*/
 char addr[30];
}stu3,stu4;

```

上述结构体可以由图 10.2 直观地表示出来。

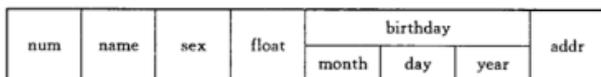


图 10.2 结构体变量成员示意图

变量 stu3 和 stu4 占用内存空间都是 71 字节。

成员名可以与程序中的变量名相同，二者不代表同一对象。程序中的 num 与 struct student 中的 num 是两回事，互不干扰。想一想这是为什么？

## 2) 结构体变量的初始化

结构体变量和其他简单类型变量一样，也可以初始化。

由于有三种定义变量的方法，变量初始化也有三种格式。

格式一：

```

struct 结构体类型名 变量名={成员 1 初值,成员 2 初值,···,成员 n 初值};

```

例如:struct student stu1={10001,"Liming",'M',18.95.0,"HubeiEnshi"};

```
struct student1 stu3={10001,"Liming",'M', 95.0,1990,5,1,"HubeiEnshi"};
```

格式二:

```
struct 结构体类型名
{
 成员列表;
}变量名={成员 1 初值,成员 2 初值,...,成员 n 初值};
```

例如:

```
struct student
{
 int num;
 char name[20];
 char sex;
 int age;
 float score;
 char addr[30];
}stu1={10001,"Liming",'M',18,95.0,"Enshi"},stu2={10002,"Wang",
'F',17,93,"YiChang"};
struct date //日期结构体
{
 int year;
 int month;
 int day;
};
struct student1
{
 int num;
 char name[20];
 char sex;
 float score;
 struct date birthday; /*birthday 是 struct date 类型*/
 char addr[30];
}stu3={10001,"Liming",'M', 95.0,1990,5,1, "Enshi"},stu4={10002,
"Wang",'F',93,1991,7,23,"YiChang"};
```

格式三:

```
struct
{
 成员列表;
}变量名={成员 1 初值,成员 2 初值,...,成员 n 初值};
```

初始化结构体变量时,一定要注意成员初始值顺序应当与成员定义的顺序一致,否则会出现错误。

### 3. 结构体变量的访问方法

定义变量的目的就是为了访问,结构体变量如何访问呢?

C语言不允许访问结构体变量。

例如:

```
scanf("%d,%s,%c,%d,%f,%s",&stu1); //错误
printf("%d,%s,%c,%d,%f,%s",stu1); //错误
```

只允许访问结构体成员变量。结构体成员变量有两种访问方式。

(1) 依据结构体变量访问结构体成员变量。

访问格式为:

结构体变量名.成员名

“.”是成员(又称为分量)运算符,它的优先级最高。

stu1.num=10001; num 左右各有一个运算符,num 先和哪个运算符结合呢?

根据优先级 num 应先和“.”左边的 stu1 结合,即将 stu1.num 看成一个整体,也可以这样理解:num 是属于 stu1 的。

(2) 依据结构体指针变量访问结构体成员变量。

指针可以指向变量,也可以指向结构体变量。指向结构体变量的指针称为结构体指针变量。定义结构体指针变量的方法与定义其他指向变量的指针一样。

例如:

```
struct student stu,*sp;
sp=&stu; //stu是结构体变量,&stu是变量stu的首地址。
```

指针变量访问格式为:

结构体指针变量名->成员名

“->”是指向运算符,它的优先级与“.”运算符的优先级相同。

sp->num=10001;与 stu1.num=10001;等价。使用指针变量访问结构体成员比使用结构体变量名更方便。

**【例 10.1】** 结构体变量和结构体指针变量访问结构体成员。

```
1 struct student
2 {
3 int num;
4 char name[20];
5 char sex;
6 int age;
7 char addr[30];
8 }stu1={10001,"Liming",'M',18,"HubeiEnshi"};
```

```

9 # include <stdio.h>
10 void main (
11 {
12 struct student* sp=&stu1;
13 printf("结构体变量访问结构体成员\n");
14 printf("NO.:%d\nName:%s\nSex:%c\nAge:%d\nAddr:%s", stu1.
 num, stu1.name, stu1.sex, stu1.age, stu1.addr);
15 printf("\n\n 结构体指针变量访问结构体成员\n");
16 printf("NO.:%d\nName:%s\nSex:%c\nAge:%d\nAddr:%s", sp->
 num, sp->name, sp->sex, sp->age, sp->addr);
17 printf("\n");
18 }

```

程序运行结果：

结构体变量访问结构体成员

```

NO.:10001
Name:Liming
Sex:M
Age:18
Addr:HubeiEnshi

```

结构体指针变量访问结构体成员

```

NO.:10001
Name:Liming
Sex:M
Age:18
Addr:HubeiEnshi

```

从上述实例可以看出,用结构体变量访问结构体成员与用结构体指针变量访问结构体成员的结果是一样的。

结构体变量、结构体指针变量与其他简单类型变量、指针变量一样,也有垂直方向上的相等关系。

|             | 地址   | 数据值 |
|-------------|------|-----|
| 结构体变量 stu:  | &stu | stu |
| 结构体指针变量 sp: | sp   | *sp |

在有定义 struct student stu, \*sp=&stu;之下, &stu 与 sp 等价, stu 与 \*sp 等价。指针变量也可以当变量使用(\*sp), num, 变量也可以当指针变量使用(&stu)->num。

**【例 10.2】** 结构体成员变量四种访问方法。

```

1 # include <string.h>
2 # include <stdio.h>
3 struct student

```

```
4 {
5 int num;
6 char name[20];
7 char sex;
8 int age;
9 float score;
10 char addr[30];
11 };
12 void main()
13 {
14 struct student stul,*sp;
15 sp=&stul;
16 stul.num=10001;
17 strcpy(stul.name,"LiMing");
18 stul.sex='M';
19 stul.age=18;
20 stul.score=89.5;
21 strcpy(stul.addr,"HubeiEnshi");
22 printf("No.:%d\nname:%s\nsex:%c\nage:%d\nscore:%.1f\n\n",stul.num,stul.name,stul.sex,stul.age,stul.score,stul.addr);
23 printf("\n");
24 printf("No.:%d\nname:%s\nsex:%c\nage:%d\nscore:%.1f\n\n",(*sp).num,(*sp).name,(*sp).sex,(*sp).age,(*sp).score,(*sp).addr);
25 printf("\n");
26 printf("No.:%d\nname:%s\nsex:%c\nage:%d\nscore:%.1f\n\n",sp->num,sp->name,sp->sex,sp->age,sp->score,sp->addr);
27 printf("\n");
28 printf("No.:%d\nname:%s\nsex:%c\nage:%d\nscore:%.1f\n\n",(&stul)->num,(&stul)->name,(&stul)->sex,(&stul)->age,(&stul)->score,(&stul)->addr);
29 }
```

程序运行结果:

```
No. :10001
name:LiMing
sex:M
age:18
```

```
score:89.5
addr:HubeiEnshi
```

```
No. :10001
name:LiMing
sex:M
age:18
score:89.5
addr:HubeiEnshi
```

```
No. :10001
name:LiMing
sex:M
age:18
score:89.5
addr:HubeiEnshi
```

```
No. :10001
name:LiMing
sex:M
age:18
score:89.5
addr:HubeiEnshi
```

从上述实例可以看出,四种访问方法的结果是一样的。

说明:

(1) 如果成员本身又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一级的成员。只能对最低的成员进行赋值或存取操作。例如:

```
stu1.birthday.year=1983;
stu1.birthday.month=06;
stu1.birthday.day=25;
```

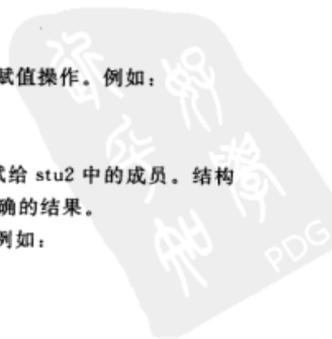
(2) 结构体变量虽然不能进行输入/输出操作,但可以进行赋值操作。例如:

```
struct student stu1,stu2;
stu2=stu1;
```

该赋值语句能将 stu1 的所有成员按结构体成员变量顺序分别赋给 stu2 中的成员。结构体变量的赋值操作,要求两变量的类型相同,否则将不能得到正确的结果。

(3) 结构体成员变量可以像普通变量一样进行各种运算。例如:

```
stu1.num++;
stu1.score=stu2.score;
```



```
stul.age+=2;
```

(4) 可以访问结构体成员变量的地址,也可以访问结构体变量的地址。例如:

```
scanf("%d",&stul.num); //输入一个整数送给结构体成员 stul.num
printf("%x",&stul); //输出结构体变量的首地址
```

#### 4. 结构体变量和结构体指针变量在函数中的应用

结构体变量和结构体指针变量在函数中有非常重要的应用,它们既可以作函数的形参,也可以作函数的实参,还可以作函数类型返回结构体。

与简单变量一样,结构体变量和成员变量和结构体指针变量都可以作函数的参数。

##### 1) 结构体成员变量作实参

这与前面介绍的简单变量作实参一样,属于“值传递”方式,只是要注意形参与实参在类型上要保持一致,如图 10.3 所示。

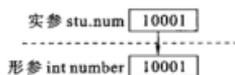


图 10.3 结构体成员变量作实参时数据传递示意图

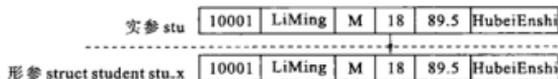


图 10.4 结构体变量作实参和形参时数据传递示意图

##### 2) 结构体变量作参数

结构体变量作实参,要求形参也是相同类型的结构体变量,参数传递采用的是“值传递”的方式,形参在函数调用期间也要占用内存单元,因此这种传递方式在空间与时间上开销较大,传递过程如图 10.4 所示。

##### 3) 结构体指针变量作参数

结构体指针变量作参数,传递的是结构体变量的首地址,要求形参也是一个能接受地址的相同类型的指针变量,地址传递过程如图 10.5 所示。

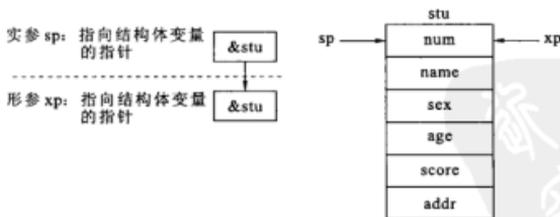


图 10.5 结构体指针变量作参数时地址传递示意图

**【例 10.3】** 有一个结构体变量 `stu`,内含学生学号、姓名和三门课的成绩,要求在 `main` 函数中赋值,在另一个函数 `output` 中将它们打印输出。

下面用三种方法实现上述程序。

## (1) 结构体成员变量作实参。

```
1 # include <stdio.h>
2 # include <string.h>
3 # define FORMAT "%d\n%s\n%.1f\n%.1f\n%.1f\n"
4 struct student
5 {
6 int num;
7 char name[20];
8 float score[3];
9 };
10
11 void main()
12 {
13 void output(int, char*, float, float, float); //函数声明
14 struct student stu;
15 stu.num=12345;
16 strcpy(stu.name, "Liming");
17 stu.score[0]=67.5f;
18 stu.score[1]=89.0f;
19 stu.score[2]=78.6f;
20 output(stu.num, stu.name, stu.score[0], stu.score[1], stu.
 score[2]); //结构体成员变量作实参
21 }
22
23 void output(int num, char * pname, float score1, float score2,
 float score3)
24 {
25 printf(FORMAT, num, pname, score1, score2, score3);
26 }
```

## 程序运行结果：

```
12345
Liming
67.5
89.0
78.6
```

## (2) 结构体变量作实参。

```
1 # include <stdio.h>
2 # include <string.h>
3 # define FORMAT "%d\n%s\n%.1f\n%.1f\n%.1f\n"
```



```
4 struct student
5 {
6 int num;
7 char name[20];
8 float score[3];
9 };
10 void main()
11 {
12 void output(struct student); //函数声明
13 struct student stu;
14 stu.num=12345;
15 strcpy(stu.name,"Liming");
16 stu.score[0]=67.5f;
17 stu.score[1]=89.0f;
18 stu.score[2]=78.6f;
19 output(stu); //结构体变量作实参
20 }
21
22 void output(struct student stu) //结构体变量名作形参
23 {
24 printf(FORMAT,stu.num,stu.name,stu.score[0],stu.
 score[1],stu.score[2]);
25 }
```

程序运行结果同上。

### (3) 结构体指针变量作参数。

```
1 # include <stdio.h>
2 # include <string.h>
3 # define FORMAT "%d\n%s\n%.1f\n%.1f\n%.1f\n"
4 struct student
5 {
6 int num;
7 char name[20];
8 float score[3];
9 };
10 void main()
11 {
12 void output(struct student*); //函数声明
13 struct student stu;
14 stu.num=12345;
```



```

15 strcpy(stu.name,"Liming");
16 stu.score[0]=67.5f;
17 stu.score[1]=89.0f;
18 stu.score[2]=78.6f;
19 output(&stu); //结构体变量地址作实参
20 }
21
22 void output(struct student* sp) //结构体指针变量作形参
23 {
24 printf(FORMAT,sp->num,sp->name,sp->score[0],sp->score
 [1],sp->score[2]);
25 }

```

程序运行结果同上。

### 10.2.3 结构体数组的应用

结构体数据也可以成为数组元素,和前面讲过的数组不一样的是,结构体数组中的元素是一个结构体类型的数据。

#### 1. 结构体数组的定义与初始化

##### 1) 结构体数组的定义

由于结构体数组元素是结构体类型,所以在定义结构体数组之前必须先定义结构体类型。如:前面定义的结构体类型 struct student。定义好结构体类型之后,就可以定义结构体数组了。定义结构体数组也有三种与定义结构体变量相似的方法。

(1) 先定义结构体类型再定义结构体数组,定义数组的方法与定义简单类型数组相似。

例如:定义好结构体类型 struct student 后,再定义有两个元素、数组名叫 stu 的数组。

```
struct student stu[2];
```

(2) 定义结构体类型同时,定义结构体数组。

```

struct student
{
 int num;
 char name[20];
 char sex;
 int age;
 float score;
 char addr[30];
}stu[2];

```



## (3) 匿名定义结构体数组。

```
struct
{
 int num;
 char name[20];
 char sex;
 int age;
 float score;
 char addr[30];
}stu[2];
```

## 2) 结构体数组的初始化

与其他类型的数组一样,对结构体数组也可以初始化,由于有三种定义数组的方法,初始化数组也有三种不同的格式。

格式一:

```
struct 结构体类型名 数组名[N]={元素 1 初值},{元素 2 初值},...,{元素 n 初值};
```

例如:

```
struct student stu[2]=
{
 {10001,"LiMing",'M',18,89.5,"HubeiEnshi"},
 {10002,"ZhangJun",'F',17,98,"HubeiYichang"}
};
```

格式二:

```
struct 结构体类型名
{
 成员列表;
}数组名[N]={元素 1 初值},{元素 2 初值},...,{元素 n 初值};
```

例如:

```
struct student
{
 int num;
 char name[20];
 char sex;
 int age;
 float score;
 char addr[30];
}stu[2]={10001,"LiMing",'M',18,89.5, "HubeiEnshi"},
```

```
 {10002,"ZhangJun",'F',17,98,"HubeiYichang"}
};
```

格式三:

```
struct
{
 成员列表;
}数组名[N]={元素 1 初值},{元素 2 初值},...,{元素 n 初值};
```

结构体数组初始化的其他特性与简单类型数组的特性相同,结构体数组也遵循数组三要素的原则。

上述结构体数组初始化后,系统在内存中为该数组开辟的空间大小为: $63 \times 2 = 126\text{B}$ ,开辟的内存空间如图 10.6 所示。

## 2. 结构体数组的访问方法

与简单类型数组一样,结构体数组的访问方法也有下标访问法和指针访问法。只是要特别注意,结构体数组元素也相当于结构体变量,对它不能进行整体输入/输出,输入/输出的是结构体数组元素中的成员变量。

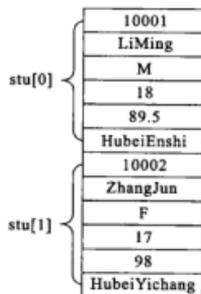


图 10.6 结构体数组在内存开辟空间示意图

### 1) 下标访问法

结构体数组的下标访问法与简单类型数组的下标访问法相似,也是用循环来实现对数组元素的访问的。

**【例 10.4】**设计有三个数组元素的学生属性数组,要求用下标访问法分别实现对学属性的输入、输出操作。

```
1 # include <stdio.h>
2 # define N 3
3 struct student
4 {
5 int num;
6 char name[20];
7 char sex;
8 int age;
9 float score;
10 char addr[30];
11 };
12 void input(struct student stu[],int);
13 void output(struct student stu[],int);
14 void main()
```

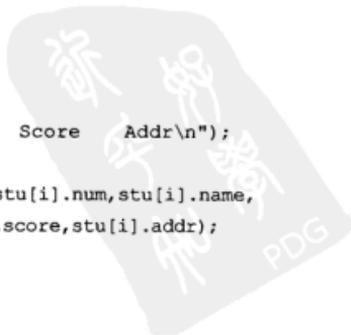


```
15 {
16 struct student stu[N];
17 input(stu,N);
18 output(stu,N);
19 }
20
21 void input(struct student stu[],int n)
22 {
23 int i;
24 for(i=0;i<n;i++)
25 {
26 printf("请输入第%d记录\n",i+1);
27 printf("学号:");
28 scanf("%d",&stu[i].num);
29 printf("姓名:");
30 scanf("%s",stu[i].name);
31 getchar(); //消除回车对输入性别的影响
32 printf("性别:");
33 scanf("%c",&stu[i].sex);
34 printf("年龄:");
35 scanf("%d",&stu[i].age);
36 printf("成绩:");
37 scanf("%f",&stu[i].score);
38 printf("地址:");
39 scanf("%s",stu[i].addr);
40 }
41 }
42
43 void output(struct student stu[],int n)
44 {
45 int i;
46 printf(" No. Name Sex Age Score Addr\n");
47 for(i=0;i<n;i++)
48 printf("%5d%10s%5c%6d%9.1f%15s\n",stu[i].num,stu[i].name,
49 stu[i].sex,stu[i].age,stu[i].score,stu[i].addr);
49 }
```

程序运行结果:

请输入第1记录

学号:10001 <回车>



姓名:LiMing <回车>  
 性别:M <回车>  
 年龄:18 <回车>  
 成绩:89.5 <回车>  
 地址:HubeiEnshi <回车>  
 请输入第 2 记录  
 学号:10002 <回车>  
 姓名:ZhangJun <回车>  
 性别:F <回车>  
 年龄:17 <回车>  
 成绩:98 <回车>  
 地址:HubeiYichang <回车>  
 请输入第 3 记录  
 学号:10003 <回车>  
 姓名:WangTong <回车>  
 性别:M <回车>  
 年龄:19 <回车>  
 成绩:87 <回车>  
 地址:HubeiWuhan <回车>

| No.   | Name     | Sex | Age | Score | Addr         |
|-------|----------|-----|-----|-------|--------------|
| 10001 | LiMing   | M   | 18  | 89.5  | HubeiEnshi   |
| 10002 | ZhangJun | F   | 17  | 98.0  | HubeiYichang |
| 10003 | WangTong | M   | 19  | 87.0  | HubeiWuhan   |

227

本实例要注意的是第 31 行,由于输入姓名后,回车符没有送给姓名字符数组,它将保留在键盘中,接着输入性别,性别类型为字符,而回车符也是字符,如果不想办法去掉这个字符,将会影响到后面字符数据的输入,这里用 `getchar` 函数来“吃掉”回车符,以保证后面字符数据的正确输入。

## 2) 指针访问法

**【例 10.5】** 用指针访问法重新设计例 10.4。

```
1 # include <stdio.h>
2 # define N 3
3 struct student
4 {
5 int num;
6 char name[20];
7 char sex;
8 int age;
9 float score;
```



```
10 char addr[30];
11 };
12 void input (struct student*,int);
13 void output (struct student*,int);
14 void main()
15 {
16 struct student stu[N], *sp;
17 sp=stu;
18 input (sp,N);
19 output (sp,N);
20 }
21
22 void input (struct student* sp,int n)
23 {
24 int i=1;
25 struct student*p=sp;
26 for (;sp<p+n;sp++)
27 {
28 printf("请输入第%d记录\n",i);
29 printf("学号:");
30 scanf("%d",&sp->num);
31 printf("姓名:");
32 scanf("%s",sp->name);
33 getchar();
34 printf("性别:");
35 scanf("%c",&sp->sex);
36 printf("年龄:");
37 scanf("%d",&sp->age);
38 printf("成绩:");
39 scanf("%f",&sp->score);
40 printf("地址:");
41 scanf("%s",sp->addr);
42 i++;
43 }
44 }
45
46 void output (struct student* sp,int n)
47 {
48 struct student*p=sp;
```



```

49 printf(" No. Name Sex Age Score Addr\n");
50 for(;sp<p+n;sp++)
51 printf("%5d%10s%5c%6d%9.1f%15s\n", sp->num, sp->name,
 sp->sex, sp->age, sp->score, sp->addr);
52 }

```

程序运行结果与例 10.4 相同。

从上述两种实例可以看出,结构体数组名也与简单类型数组名一样也可以作实参和形参在函数之间传递地址。

## 10.3 扩展知识与理论

### 10.3.1 结构体变量作函数类型

结构体变量和其他简单变量一样也可以作函数类型,返回结构体数据,这样可以大大方便程序设计。

**【例 10.6】** 将例 10.3 作如下修改,学生数据输入用 input 函数实现,output 函数将其输出,主函数实现输入/输出函数调用。

```

1 # include <stdio.h>
2 # include <string.h>
3 # define FORMAT "学号:%d\n 姓名:%s\n 成绩 1:%.1f\n 成绩 2:%.1f\n 成绩 3:%.1f\n"
4 struct student input(); //函数声明
5 void output(struct student*); //函数声明
6 struct student
7 {
8 int num;
9 char name[20];
10 float score[3];
11 };
12
13 void main()
14 {
15 struct student stu,*sp;
16 sp=&stu;
17 stu=input(); //调用输入函数
18 printf("\n 输出学生信息\n");
19 output(sp); //调用输出函数
20 }
21

```



```
22 struct student input ()
23 {
24 struct student stu,*sp;
25 int i;
26 sp=&stu;
27 printf("请输入学号:");
28 scanf("%d",&sp->num);
29 printf("请输入姓名:");
30 scanf("%s",sp->name);
31 for(i=0;i<3;i++)
32 {
33 printf("请输入第%d门成绩:",i+1);
34 scanf("%f",&sp->score[i]);
35 }
36 return stu;
37 }
38
39 void output(struct student* sp) //结构体指针变量作形参
40 {
41 printf(FORMAT,sp->num,sp->name,sp->score[0],sp->score
42 [1],sp->score[2]);
43 }
```

程序运行结果:

```
请输入学号:1001<回车>
请输入姓名:LiMing<回车>
请输入第1门成绩:98<回车>
请输入第2门成绩:78<回车>
请输入第3门成绩:85<回车>
```

输出学生信息

```
学号:1001
姓名:LiMing
成绩 1:98
成绩 2:78
成绩 3:85
```

通过上面的实例可以看出,函数声明可以放在函数内,也可以放在函数外,但一定要在调用该函数之前。`sp->num`表示的是成员变量,取它的地址表示成`&sp->num`,此外取数组元素地址也要表示成`&sp->score[i]`。想一想,在上例中输入姓名时,为什么使用的是`sp->name`,而不是`&sp->name`?

### 10.3.2 共用体类型的应用

设计一个 people 结构体,用来存放学生或教师信息,设计如下:

| 姓名 | 编号 | 性别 | 系别 | 职业 | 班级/职称 |    |
|----|----|----|----|----|-------|----|
|    |    |    |    |    | 班级    | 职称 |
|    |    |    |    |    |       |    |

学生和教师信息中前五项都可以相同(类型),但最后一项就不同了。如果是学生就要填写整型的班级号,如果是教师就要填写字符串型的职称了。

怎么具体实现这一结构体呢,如果这样定义:

```
struct people
{
 ...
 int class;
 char office[10];
};
```

很显然,在某种具体情况下,一个 people 变量只能是学生或者是教师,不可能两样都是。所以,是学生给出班级号,是教师给出职称,不会两样同时都存在。这就造成两个成员在同一时刻只使用一个成员变量的情况,另一个成员变量则空闲了,从而浪费了内存空间。如果定义的变量较多,造成的内存浪费就越大,这不是一个好办法。

造成上述浪费的根本原因是因为无论班级号还是职称都有各自独立的存储空间。

怎么解决上述问题,能否做到是学生只分配班级号空间,是教师就分配职称空间?下面介绍的共用体可以基本做到这一点,以达到节省内存空间和更符合实际的目的。

#### 1. 共用体类型与共用体变量的定义

有时需要将几种不同类型的变量存放同一段内存单元中。这些变量怎么占用同一单元呢,有以下几个因素要考虑:

- ① 这些不同变量的起始地址都相同;
- ② 每次只能存放其中一个变量,即使用覆盖技术,几个变量互相覆盖;
- ③ 共用体变量所占内存大小,取几个不同类型变量所占内存中最大的那一个。

##### 1) 共用体类型的定义

这种使几个不同类型的变量共占同一段内存的结构,称为“共用体”类型。共用体类型是用户自定义类型,它也要遵循先定义类型、再定义变量的原则。

类型定义格式为:

```
union 共用体名
{
 成员列表
};
```

其中,union 为定义共用体类型的关键字。

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

## 2) 共用体变量的定义

定义了共用体类型之后,就可以根据共用体类型定义共用体变量了。定义共用体变量的方法与定义结构体变量的方法相似,也有三种定义方法。

(1) 定义类型与定义变量分开进行。

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

(2) 定义类型的同时定义变量。

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

(3) 匿名定义。

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

想一想,“a,b,c”共用体变量各占多大的存储空间? 是  $4+1+4=9\text{B}$ , 还是取最大的成员所占的空间  $4\text{B}$ ?

## 2. 共用体成员变量的访问方法

共用体变量的访问方法同结构体变量的访问方法一样,即不能整体访问共用体变量,只能引用共用体成员变量。访问方法也有根据变量名访问法和根据指针变量名访问法两种。

根据变量名访问格式为：

共用体变量名.成员名

根据指针变量名访问格式为：

共用体指针变量名->成员名

例如：`printf("%d",a);` //错误

只能访问共用体成员变量：`a.i`；或 `b.ch`；或 `c.f`；。

如果 `p` 是指向共用体的指针变量，也可以这样引用：`p->i`；或 `p->ch`；或 `p->f`；。

**【例 10.7】** 用共用体变量和共用体指针变量访问共用体成员变量。

```

1 # include <stdio.h>
2 union data
3 {
4 int i;
5 char ch;
6 float f;
7 };
8 void main()
9 {
10 union data a,b,c,*pa,*pb,*pc;
11 a.ch='A';
12 b.f=98.0;
13 c.i=34;
14 pa=&a;
15 pb=&b;
16 pc=&c;
17 printf("a.ch=%c,b.f=%.1f,c.i=%d\n",a.ch,b.f,c.i);
18 printf("pa->ch=%c,pb->f=%.1f,pc->i=%d\n",pa->ch,pb->
 f,pc->i);
19 }

```

程序运行结果：

```

a.ch=A,b.f=98.0,c.i=34
pa->ch=A,pb->f=98.0,pc->i=34

```

与结构体变量一样，共用体变量也可以作函数参数和函数类型在函数之间传递数据。

**【例 10.8】** 共用体变量作函数参数和函数类型在函数之间传递数据。

```

1 # include <stdio.h>
2 union data
3 {
4 int i;

```

```

5 char ch;
6 float f;
7 };
8 union data input ();
9 void output (union data);
10 void main ()
11 {
12 union data a;
13 a=input ();
14 output (a);
15 }
16
17 union data input () //共用体类型作函数类型
18 {
19 union data a;
20 a.i=23;
21 a.ch='A';
22 return a; //返回共用体变量
23 }
24
25 void output (union data a) //共用体变量作函数参数
26 {
27 printf ("%d\n",a.i);
28 }

```

程序运行结果：

```
65
```

说明：

(1) 同一个内存段可以用来存放几种不同类型的成员,但在某一时刻只能存放其中一种成员变量,而不能同时存放几种成员变量。

(2) 内存中存放的数据是最后一次存入的内容,在此之前的内容全部被冲掉(覆盖),因此起作用的成员是最后一次存放的成员。例 10.8 的 20 行 21 行就清楚地说明了这一点。

(3) 共用体和结构体可以互相嵌套定义。

**【例 10.9】** 设有若干个人的数据,其中有学生和教师。学生的数据中包括:姓名、学号、性别、职业、班级。教师的数据包括:姓名、号码、性别、职业、职务或职称,现在要将它们放在统一的表格中。如果“job”项为“s”,则第 5 项为 class,如果“job”项为“t”,则第 5 项为 office。显然,对第 5 项要处理成共用体的形式将 class 和 office 放在同一段内存中,如图 10.7 所示。

| 姓名     | 编号    | 性别 | 职业 | 班级/职称 |      |
|--------|-------|----|----|-------|------|
|        |       |    |    | 班级    | 职称   |
| LiMing | 10001 | M  | s  | 501   |      |
| WangXi | 20001 | F  | t  |       | prof |

图 10.7 学生与教师信息示意图

要求输入人员的相关数据,然后再输出,程序流程图如图 10.8 所示。

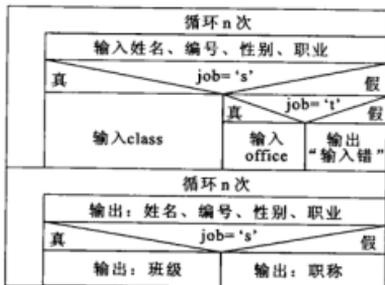


图 10.8 例 10.10 的程序流程 N-S 图

```

1 # include <stdio.h>
2 # include <stdlib.h>
3 # define N 2
4 union option
5 {
6 int classa;
7 char office[10];
8 };
9 struct people
10 {
11 int num;
12 char name[10];
13 char sex;
14 char job;
15 union option category; //共用体成员变量
16 };
17 void input(struct people*,int);
18 void output(struct people*,int);
19 void main()

```

```
20 {
21 struct people person[N], *sp;
22 sp=person;
23 input(sp,N);
24 output(sp,N);
25 }
26
27 void input(struct people* sp,int n)
28 {
29 struct people*p=sp;
30 char numstr[20];
31 for(;sp<p+n;sp++)
32 {
33 printf("请输入编号:");
34 gets(numstr);
35 sp->num=atoi(numstr);
36 printf("请输入姓名:");
37 scanf("%s",sp->name);
38 getchar(); /*吃掉输入结束后的"回车"符,下同*/
38 printf("请输入性别(M/F):");
40 scanf("%c",&sp->sex);
41 getchar();
42 printf("请输入职业(t/s):");
43 sp->job=getchar();
44 if(sp->job=='s')
45 {
46 printf("请输入班级号:");
47 scanf("%d",&sp->category.classa);
48 getchar();
49 }
50 else
51 if(sp->job=='t')
52 {
53 printf("请输入职称:");
54 scanf("%s",sp->category.office);
55 getchar();
56 }
57 else
58 printf("input error!");
```



```

59 }
60 }
61
62 void output(struct people* sp,int n)
63 {
64 struct people* p=sp;
65 printf(" NO. Name Sex Job Class/Office\n");
66 for (;sp<p+n;sp++)
67 {
68 printf("%-10d%-11s",sp->num,sp->name);
69 if (sp->sex=='M' ||sp->sex=='m')
70 printf("男");
71 if (sp->sex=='F' ||sp->sex=='f')
72 printf("女");
73 if (sp->job=='s')
74 printf("\t\t 学生");
75 if (sp->job=='t')
76 printf("\t\t 教师");
77 if (sp->job=='s')
78 printf("\t\t%d\n",sp->category.classa);
79 if (sp->job=='t')
80 printf("\t\t%s\n",sp->category.office);
81 }
82 }

```

程序运行结果:

请输入编号:10001 <回车>  
 请输入姓名:LiMing <回车>  
 请输入性别 (M/F):M <回车>  
 请输入职业 (t/s):s <回车>  
 请输入班级号:501 <回车>  
 请输入编号:20001 <回车>  
 请输入姓名:WangXi <回车>  
 请输入性别 (M/F):F <回车>  
 请输入职业 (t/s):t <回车>  
 请输入班级号:教授<回车>

| NO.   | Name   | Sex | Job | Class/Office |
|-------|--------|-----|-----|--------------|
| 10001 | LiMing | 男   | 学生  | 501          |
| 20001 | WangXi | 女   | 教师  | 教授           |



### 10.3.3 枚举类型的应用

所谓“枚举”是指变量可能的取值的个数较少,可以一一列举出来,变量的值只限于列举出来的值的范围内。这种情况很多,如一周七天、一年十二个月、七种基本颜色、逻辑类型的“真”和“假”。而有些数据是不能枚举的,如整型、实型数据,它们都是连续而不可枚举的。

#### 1. 枚举类型与枚举变量的定义

枚举也需要先定义类型再定义变量。实际上枚举类型并不是构造类型,它是一种与整型兼容的数据类型。把它放在这里介绍是基于使用枚举类型也要先定义类型再定义变量的考虑。

##### 1) 枚举类型的定义

定义格式为:

```
enum 枚举类型名
{
 取值列表
};
```

其中,enum为枚举类型关键字。

例如:定义一周七天的枚举类型。

```
enum WEEKDAY{Sun,Mon,Tue,Wed,Thu,Fri,Sat};
```

##### 2) 枚举变量的定义

定义枚举类型的变量和前面定义结构体及共用体的变量相似,也可以采用三种方法。

(1) 先定义枚举类型再定义枚举变量。例如:

```
enum WEEKDAY today, tomorrow;
```

(2) 定义枚举类型的同时定义枚举变量。例如:

```
enum WEEKDAY
{
 Sun,Mon,Tue,Wed,Thu,Fri,Sat
}today,tomorrow;
```

(3) 匿名定义。例如:

```
enum
{
 Sun,Mon,Tue,Wed,Thu,Fri,Sat
}today,tomorrow;
```

枚举类型中的取值列表称为枚举元素或枚举常量,它们是用户定义的标识符,一般为了与变量区别可以用大写字母。同时,定义的枚举常量不会自动地代表什么含义,例如,

定义 Sun 不会自动代表“星期天”，用什么标识符代表什么含义完全由程序员自己决定，需要在程序中作相应的处理。

## 2. 枚举类型与整型的关系和说明

枚举类型的“取值列表”中的符号与整数有一一对应关系，每个符号依次与 0,1,2,3, … 对应，如前面的枚举类型定义中符号与整数有如下对应关系。

```
Sun—0 Mon—1 Tue—2 Wed—3 Thu—4 Fri—5 Sat—6
```

也就是说枚举类型中定义的“符号”实际上不是符号，归根结底是整数。当然也可以任意改变其对应关系。例如：

```
enum WEEKDAY{Sun=0,Mon,Tue=5,Wed,Thu,Fri,Sat};
```

此时，有：

```
Sun—0 Mon—1 Tue—5 Wed—6 Thu—7 Fri—8 Sat—9
```

所以枚举值是可以进行算术运算的，Sun+2 是允许的。但是 Sun=2 操作是非法的，因为 Sun 是常量，不能用赋值符来改变其值。

**【例 10.10】** 根据今天求明天是星期几。

```
1 #include <stdio.h>
2 int nextDayOf(enum WEEKDAY);
3 enum WEEKDAY{Sun,Mon,Tue,Wed,Thu,Fri,Sat};
4 void main()
5 {
6 enum WEEKDAY weekday;
7 int tomorrow;
8 weekday=Tue;
9 tomorrow=nextDayOf(weekday);
10 switch(tomorrow)
11 {
12 case 0:printf("Sunday\n");
13 break;
14 case 1:printf("Monday\n");
15 break;
16 case 2:printf("Tuesday\n");
17 break;
18 case 3:printf("Wednesday\n");
19 break;
20 case 4:printf("Thursday\n");
21 break;
22 case 5:printf("Friday\n");
23 break;
24 case 6:printf("Saturday\n");
```



```

25 break;
26 default:printf("Parameter error\n");
27 }
28 }
29
30 int nextDayOf(WEEKDAY today)
31 {
32 if(today==Sat)
33 return Sun;
34 else
35 return today+1;
36 }

```

程序运行结果：

Wednesday

说明：

- (1) 枚举元素是常量不是变量。
- (2) C 语言的输出函数 printf() 不识别枚举符号，因此不能用输出函数得到枚举符号，如：

```

enum WEEKDAY today= Tue;
printf("%d", today); //只能得到 2,而不能得到 Tue.

```

- (3) 枚举值可以进行判断比较，比较时按枚举值的顺序号进行。例如：  
today=Mon; 则 today> Sun 比较结果为“真”。

### 10.3.4 常见错误及处理方法

常见错误 1: 直接使用结构体变量进行输入输出操作。

C 语言规定，结构体变量不能进行直接输入输出操作，只能对结构体变量成员进行输入输出操作，结构体变量可以进行赋值操作。

常见错误 2: 结构体成员变量输入操作时地址引用不当。例如：

```

scanf("%d", stu.num);或 scanf("%d", sp->num); //错误
scanf("%s", &stu.name);或 scanf("%s", &sp->name); //错误

```

结构体成员变量也是变量，它具有变量的全部属性，变量取地址操作要用到取地址符“&”。stu.name 和 sp->name 是数组名，数组名代表数组的首地址，因此不必再加取地址符。正确的语句应当是：

```

scanf("%d", &stu.num);或 scanf("%d", &sp->num); //正确
scanf("%s", stu.name);或 scanf("%s", sp->name); //正确

```

常见错误 3: 结构体类型定义时不恰当地使用了递归定义。

C 语言规定，定义结构体类型时允许嵌套定义，但不允许除自身指针类型之外的递归定义。所谓嵌套定义是指结构体成员变量是一个已定义好的其他结构体类型，所谓递归定义是指用自身类型来定义结构体成员变量。

下面的递归定义是不允许的。

```
struct student
{
 int num;
 char name[20];
 float score[3];
 struct student next; //变量递归定义
};
```

这是因为不能准确地得到该类型为变量开辟空间所设计的模型大小。即用这种结构体类型定义变量后,不知道内存为变量到底分配了多大的存储空间。如果将其换成指针类型时,这样的递归又是允许的。

```
struct student
{
 int num;
 char name[20];
 float score[3];
 struct student*next; //指针变量递归定义
};
```

这是因为,对于任何类型的指针变量,内存总是为它分配 4 个字节的存储空间,这样就能清楚地知道用这种类型定义变量后,内存开辟空间的大小。上述类型定义变量后内存为每个变量分配 40 字节的存储空间。

常见错误 4:给共用体变量初始化,给共用体变量赋值,直接访问共用体变量。

```
union data
{
 int i;
 char ch;
 float f;
};
union data a={12, 'c',12.5}; /*不能初始化共用体变量*/
a=12; /*不能对共用体变量赋值*/
printf("a=%c\n",a); /*不能直接访问共用体变量*/
```

## 10.4 深入训练

(1) 自定义一个结构体类型的变量,其成员包括学号、姓名、年龄、性别,并将其类型声明为 STUDENT,然后用该类型定义一个学生类型的变量,进行赋值操作,并输出其值。

(2) 将上述程序改为四个学生,用结构体数组实现输入输出学生的基本信息,要求每

行输出一个学生记录。

(3) 利用结构体类型编写一程序,实现输入一个学生的数学期中和期末成绩,然后计算并输出平均成绩。

(4) 用户手机通信收费由月租费、通话费和短信费组成,编写程序分别输入两个人的各项费用,计算两个人的通信费之差。

(5) 设有N名考生,每个考生的数据包括考生号、性别和成绩,编写一程序,要求用指针访问方法找出所有女生的成绩和成绩最好的男生,并输出他们的信息。

(6) 设计一枚枚举类型的月份,编写一个显示下一个月名称的函数 next\_month(),在主函数中实现输入一个月份,输出下一个月份。

## 习 题 10

### 10.1 填空题

(1) 构造类型要先定义\_\_\_\_\_,再定义\_\_\_\_\_。

(2) 定义结构体类型的关键字是\_\_\_\_\_,定义共用体类型的关键字是\_\_\_\_\_。

(3) 结构体变量占用内存空间的大小是\_\_\_\_\_,共用体变量占用内存空间的大小由\_\_\_\_\_决定。

(4) 结构体定义中,其成员类型可以是除\_\_\_\_\_任何已有类型,也可以是\_\_\_\_\_的指针类型,也就是说结构体不允许\_\_\_\_\_定义。

(5) 下面程序的正确输出结果为\_\_\_\_\_。

```
include <stdio.h>
void main()
{
 struct
 {
 int num;
 float score;
 }person;
 int num;
 float score;
 num=1;
 score=2;
 person.num=3;
 person.score=5;
 printf("%d,%f",num,score);
}
```

(6) 动态开辟存储空间的函数原型是\_\_\_\_\_,动态释放存储空间的函数原型是\_\_\_\_\_。

## 10.2 判断题(判断下列叙述的正确性,正确的请打“√”,错误的请打“×”)

- (1) 结构体只能处理不同类型的数组,相同类型的数组不能处理。 ( )
- (2) 结构体变量不能进行整体的输入/输出操作,但可以进行赋值操作。 ( )
- (3) 结构体类型不能进行嵌套定义。 ( )
- (4) 结构体变量可以作形参也可以作实参,但不能作返回值。 ( )
- (5) 共用体与结构体除在内存中占用空间的方式不同外,其他访问方法都相似。 ( )
- (6) 枚举类型元素本质上是整型常量。 ( )

## 10.3 选择题

(1) 当说明一个结构体变量时系统分配给它的内存是( )。

- A. 各成员所需内存量的总和
- B. 结构体中第一个成员所需内存量
- C. 成员中占内存量最大者所需的容量
- D. 结构体中最后一个成员所需内存量

(2) 计算出下列每个结构体类型定义的变量所占内存的大小:①( );②( );

③( )。

①struct AA

```
{
 int*a;
};
```

②struct BB

```
{
 int a;
 float b;
};
```

③struct CC

```
{
 char*data;
 struct BB s;
 struct CC*link;
};
```

(3) 有关结构体的正确描述是( )。

- A. 结构体成员必须是同一数据类型
- B. 结构体成员只能是不同数据类型
- C. 成员运算符“.”和“->”作用是等价的
- D. 成员就是数组元素

(4) 设有以下说明语句,则下面的叙述不正确的是( )。

```
struct student
{
 int a;
 float b;
}stutype;
```

- A. struct 是结构体类型的关键字
- B. struct student 是用户定义的结构体类型
- C. stutype 是用户定义的结构体类型名
- D. a 和 b 都是结构体成员名

(5) 以下程序的运行结果是( )。

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



```
main()
{
 struct date
 {
 int year,month,day;
 }today;
 printf("%d\n",sizeof(struct date));
}
```

- A. 2                      B. 5                      C. 6                      D. 12

(6) 当说明一个共用体时系统分配给它的内存是( )。

- A. 各成员所需内存量的总和  
B. 结构中第一个成员所需内存量  
C. 成员中占内存量最大者所需的容量  
D. 结构中最后一个成员所需内存量

(7) 下面程序运行的正确结果是( )。

```
include <stdio.h>
union opn
{
 int k;
 char ch[2];
}x;
void main()
{
 x.ch[0]=30;
 x.ch[1]=0;
 printf("%d\n",x.k);
}
```

- A. 209                      B. 208                      C. 31                      D. 30



## 单元 11 项目中学生数据的存储与重用

### 能力目标

- 能使用 `fopen` 函数和 `fclose` 函数正确打开和关闭文件。
- 能用字符读写函数读写字符数据,能用数据块读写函数读写数据。
- 能用文件定位函数对文件进行正确的定位操作,能用文件定位函数设置文件指针的当前位置,能用文件操作函数获得当前文件指针的位置。
- 能用字符串读写函数读写字符串数据,能用格式化读写函数读写数据。
- 能保存和读取“班级学生成绩管理系统”中的学生信息。

### 知识目标

- 理解文件的概念、作用和文件类型。
- 理解文件指针的概念和定义方法,理解文件指针在打开文件和关闭文件时与文件的关系。
- 正确理解十二种“文件打开方式”的含义。
- 理解并掌握字符、字符串、数据块、格式化文件读写函数的格式。
- 理解文件定位函数的功能与使用方法。

### 学习提示

数据的磁盘存储与文件操作是计算机操作中最为重要的工作,不能存储和重复使用的数据是没有意义的数据。通过学习应当掌握文件的概念,特别要理解键盘输入和显示器输出与磁盘读和写的联系与区别;掌握文件的打开、关闭方法;标准文件操作函数,文件的顺序读写与随机读写的实现方法。

## 11.1 任务 15: 项目中学生数据的存储和重复使用

“班级学生成绩管理系统”项目中数据的存储主要涉及学生信息的保存和学生信息的重复使用。学生信息的保存用 `Save` 函数,学生信息文件的打开用 `Open` 函数实现。

(1) 首先将两个函数声明修改为:

```
void Save(struct student stu[],int size); //保存文件
void Open(struct student stu[],int* size); //打开文件
```

(2) 主函数中两个函数调用修改为:

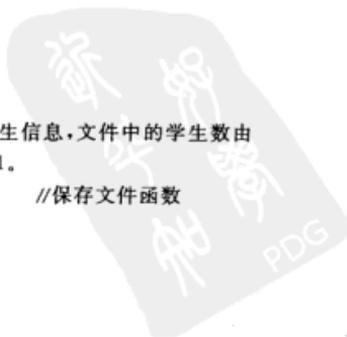
```
Open(stu,&stunum);
Save(stu,stunum);
```

(3) 两个函数定义修改为:

```
void Open(struct student stu[],int* size)//打开文件函数
{
 int i=0;
 FILE* fp;
 system("cls");
 if((fp=fopen("stuscore","rb"))==NULL)
 {
 printf("文件不能正常打开!\n");
 exit(0);
 }
 else
 {
 while(!feof(fp))
 {
 fread(&stu[i],sizeof(struct student),1,fp);
 i++;
 }
 fclose(fp);
 }
 gotoxy(25,5);
 printf("打开文件成功!");
 *size=i-1;//文件中学生数
 getch();
}
```

该函数用读数据块函数打开保存在 stuscore 文件中的学生信息,文件中的学生数由形参指针变量 size 带回。想一想,带回的学生数为什么是 i-1。

```
void Save(struct student stu[],int size) //保存文件函数
{
 FILE* fp;
 int i;
 system("cls");
 if((fp=fopen("stuscore","wb"))==NULL)
```



```

{
 printf("文件不能正常打开!\n");
 exit(0);
}
else
{
 for(i=0;i<size;i++)
 {
 fwrite(&stu[i],sizeof(struct student),1,fp);
 }
 fclose(fp);
}
gotoxy(25,5);
printf("保存文件成功,按任意键返回上级菜单!");
getch();
}

```

学生信息保存在一个名为 stuscore 的文件中,以二进制文件的形式保存,保存的学生数由形参 size 决定,用写数据块函数实现保存,保存文件成功后给出提示。

单元能力训练任务分别如下。

任务 A:能用 if-else 语句实现文本文件、二进制文件的打开与关闭操作。

任务 B:能用 if-else 语句实现追加文本文件、追加二进制文件的打开与关闭操作。

任务 C:编写用读写数据块函数实现“班级学生成绩管理系统”中数据的读写函数,并正确实现调用操作。

任务 D:模仿任务 15 实现“学生通讯录”信息的保存和重新使用操作。

## 11.2 必备知识与理论

### 11.2.1 文件的概念

文件是程序设计中一个重要的概念。所谓“文件”一般指存储在外部介质上的数据集合。C 语言把文件的概念扩大化了,它将一些设备也当作文件来处理,这样就使程序设计更加具有灵活性和通用性。这一点很重要,就是说 C 语言不仅仅将数据的集合当成文件,它还可以是设备,如键盘、显示器和打印机等。

一般而言,我们理解的文件还是指存储在外部介质上数据的集合。操作系统是以文件为单位对数据进行管理的,也就是说,如果想找到存储在外部介质上的数据,必须先按文件名找到所指定的文件,然后再从该文件中读取数据。向外部介质上存储数据也必须先建立一个文件,才能向它输出(写)数据。

从键盘上输入的数据是存放在内存中,显示器可以将内存中的数据显示输出。文件保存在外部介质中,“读”文件操作就是将磁盘文件输入到内存中,“写”文件操作就是将内

存中的数据输出到外部介质中。这一点希望读者能认真理解和体会。它是学习文件这一章面临的首要问题。

C 语言将文件看成是存储在外部介质中的字符集。根据数据在外部介质中存储的不同方式,C 语言的文件又分为 ASCII 文件和二进制文件。

ASCII 文件:又称为文本文件(txt)文件,特点是数据在外部介质中一个字节存放一个 ASCII 码字符。

二进制文件:把内存中的数据按其在内存中的存储形式原样输出到外部介质上存放。

例如,如果要存放整数 12345,整数在内存中是占两个字节的,12345 作为文本文件和作为二进制文件在内存中存放有着较大的区别。

$$(12345)_{10} = (11000000111001)_2$$

|          |          |          |          |          |          |
|----------|----------|----------|----------|----------|----------|
| ASCII 形式 | 00110001 | 00110010 | 00110011 | 00110100 | 00110101 |
| 二进制形式    | 00110000 | 00111001 |          |          |          |

从上面可以清楚地看到,以 ASCII 形式存放需要占用 5 个字节,以二进制形式只需要 2 个字节。

因此,一个 C 文件是一个字节流或二进制流。所谓“流”是一系列字节,它可以从一种类型的设备流向另一种类型的设备。就文件流来说,通过流连接起来的两个设备是计算机的内存和磁盘上的文件。

在 C 语言中对文件的存取是以字符(字节)为单位的,输入/输出数据的开始和结束仅受程序控制而不受物理符号(如回车换行符)控制。也就是说,在输出时不会自动增加回车换行符以作为结束的标志,输入时不以回车换行符为记录的间隔。我们把这种文件称为流式文件。

C 语言处理文件的方法是采用“缓冲文件系统”的方式。所谓缓冲文件的方式是指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。从内存向磁盘输出数据必须先送到内存中的缓冲区,装满缓冲区后才一起送到磁盘。如果从磁盘向内存读入数据,则从磁盘文件中先将一批数据输入到内存缓冲区,然后再从缓冲区逐个地将数据送到程序数据区,如图 11.1 所示。

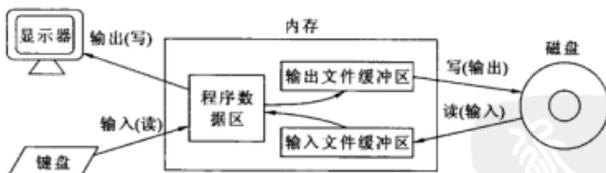


图 11.1 键盘、显示器输入/输出与磁盘文件读写示意图

## 11.2.2 文件的基本操作

### 1. 文件的打开与关闭

文件在使用(读/写)之前必须先“打开”该文件,在使用结束之后应当关闭该文件。

## 1) 文件的打开 (fopen 函数)

C 语言用 fopen() 函数来实现打开文件。fopen 函数的调用格式如下：

```
FILE *fp;
fp=fopen(文件名,使用文件方式);
```

其中,FILE 为系统定义的有关文件信息的结构体类型,有关具体内容可以查看相关书籍,这里不再叙述。\*fp 是指向文件的指针变量。

fopen 函数有下面两个参数。

文件名:被打开文件的名字。

使用文件方式:包括对文件是读还是写等(见表 11.1)。

表 11.1 文件使用方式表

| 文件使用方式    | 含 义             |
|-----------|-----------------|
| "r"(只读)   | 为输入打开一个文本文件     |
| "w"(只写)   | 为输出打开一个文本文件     |
| "a"(追加)   | 向文件尾增加数据        |
| "rb"(只读)  | 为输入打开一个二进制文件    |
| "wb"(只写)  | 为输出打开一个二进制文件    |
| "ab"(追加)  | 向二进制文件尾增加数据     |
| "r+"(读写)  | 为读/写打开一个文本文件    |
| "w+"(读写)  | 为读/写建立一个新的文本文件  |
| "a+"(读写)  | 为读/写打开一个文本文件    |
| "rb+"(读写) | 为读/写打开一个二进制文件   |
| "wb+"(读写) | 为读/写建立一个新的二进制文件 |
| "ab+"(读写) | 为读/写打开一个二进制文件   |

249

打开文件就是建立文件指针与文件的联系,即建立文件指针与文件名的对应关系。

想一想,fopen 函数的返回值是什么? 返回值是文件首地址,显然 fp 是指向了文件的首地址的文件指针。例如:

```
fp=fopen("a1","r");
```

被打开的文件名叫 a1,并且以“读”方式打开。

fp 指向文件的首地址,如图 11.2 所示。

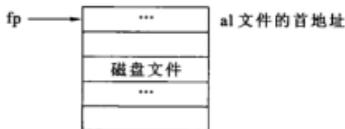


图 11.2 文件指针变量 fp 指向文件首地址示意图

打开一个文件时,系统得到三个方面的信息:

- ① 需要打开的文件名,也就是准备访问文件的名字;
- ② 使用文件的方式(是“读”还是“写”);
- ③ 让哪一个指针变量指向被打开的文件。

说明:

(1) 用“r”方式打开文件(无论是“r”、“rb”、“r+”、还是“rb+”),用于“读”,即磁盘文件输入到内存中。

(2) 用“w”方式打开文件(无论是“w”、“wb”、“w+”、还是“wb+”),用于“写”,即内存数据输出到磁盘文件中。打开文件时,如果原来不存在该文件,则在打开时新建一个以指定名字命名的文件,如果原来已存在一个以该文件名命名的文件,则在打开时将该文件删去,然后重新建立一个以该名字命名的新文件。

(3) 凡带“+”号的打开方式,打开文件时总是既能“读”,又能“写”。

(4) “a”方式打开文件时,位置指针移到文件末尾,可以实现追加操作。

(5) “b”方式打开文件时,可以对二进制文件进行操作。

(6) 如果以“r”方式打开一个并不存在的文件,或在磁盘损坏、磁盘空间不足等情况下打开文件,都会使打开文件失败。此时 fopen 函数将返回一个空指针 NULL,所以常用下面的方法打开一个文件。

```
if ((fp = fopen("file1", "r")) == NULL)
{
 printf("cannot open this file\n");
 exit(0); //退出程序,返回系统库函数
}
}
```

打开文件流程图如图 11.3 所示。

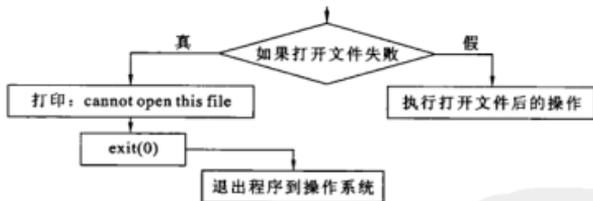


图 11.3 打开文件流程图

## 2) 文件的关闭(fclose 函数)

一个文件使用完后应该及时关闭它,以防止它再被误用。关闭文件操作完成下面两件事。

(1) 关闭文件缓冲区,将还没有装满的缓冲区数据输出到磁盘文件中,保证数据不丢失。

(2) “关闭”就是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”,

释放文件指针变量。

关闭文件的格式为：

```
fclose(文件指针);
```

例如：`fclose(fp)`；

前面曾把打开文件时所带回的指针赋给了 `fp`，现在通过 `fp` 把该文件关闭，即 `fp` 不再指向该文件。

`fclose` 函数也带回一个值，如果顺利地执行了关闭操作，则返回值为 0；否则返回 `EOF(-1)`。

## 2. 文件的读/写方法

文件打开之后，就可以对它进行读/写了。通常，C 语言的文件读/写函数是成对出现的，即有读就有写。下面介绍如下。

### 1) 读/写字符函数

(1) 读字符函数 `fgetc/getc`。

从指定的文件输入一个字符到内存。该文件必须是以读或读写方式打开的。

函数原型为：

```
int fgetc (FILE *fp);
int getc (FILE *fp);
```

`fp` 是一个文件类型指针，它指向要读的文件。

如果正常返回，则返回读取的字符代码。否则返回 `EOF`；如果读到文件结束符（‘Z’）时，也返回 `EOF`。

函数的一般用法：`ch=fgetc(fp)`；

上述语句把读入字符的 ASCII 值存入变量 `ch`。

C 语言规定，`stdin` 代表标准输入设备文件（如键盘），前面已经说过，C 语言将标准设备也看成是文件，除此之外还有标准输出设备文件 `stdout`（代表显示器）和标准出错输出文件 `stderr`。这三种文件系统自动定义、自动打开、自动关闭。实际上 `stdin`、`stdout`、`stderr` 就是这三个文件的指针。

用 `fgetc(stdin)` 输入字符与用单个字符输入函数 `getchar()` 的功能相同。

(2) 写字符函数 `fputc`。

把一个字符从内存输出到磁盘文件中。

函数原型为：

```
int fputc (char ch, FILE *fp);
int putc (char ch, FILE *fp);
```

其中，`ch` 是要写的字符，它可以是字符常量，也可以是字符变量。`fp` 是文件指针变量，指向当前打开的文件。

函数是将字符 ch 输出到 fp 所指向的文件中。如果调用成功,返回值就是输出的字符(ASCII 码),如果失败返回一个 EOF。

想一想,单个字符输入函数 putchar(ch)与 fputc(ch, stdout)功能相同吗?为什么?

向文件写入一个字符后,文件的读写位置后移一个字节,因为文件指针在读写过程中是移动的,所以字符不会重叠。

**【例 11.1】** 从键盘上输入一些字符,逐个把它们送到磁盘上去,“#”结束输入,然后输出这些字符。

```

1 # include <stdio.h>
2 # include <stdlib.h>
3 void main()
4 {
5 FILE* fp;
6 char ch, filename[10];
7 printf("请输入保存文件名:");
8 scanf("%s", filename);
9 ch=getchar(); //该语句用来吃掉字符串后的回车符
10 if((fp=fopen(filename, "w"))==NULL)
11 /*创建文件用于输出*/
12 {
13 printf("can not creat the file.\n");
14 exit(0);
15 }
16 printf("文件创建成功! 请输入字符:");
17 ch=getchar(); /*输入第一个字符*/
18 while(ch!='#')
19 {
20 fputc(ch, fp); /*把变量 ch 写入文件*/
21 putchar(ch); /*屏幕显示 ch 的值*/
22 ch=getchar(); /*继续输入下一个字符*/
23 }
24 fclose(fp); /*关闭文件*/

```

程序运行结果:

请输入保存文件名: flie1<回车>

文件创建成功! 请输入字符: One World One Deram# <回车>

One World One Deram

该程序从 16 行到 22 行可以优化为,

```

while((ch=getchar())!='#')
{

```

```
 fputc(ch,fp);
 putchar(ch);
}
```

**【例 11.2】** 用读/写字符函数实现两文本文件的复制。

```
1 # include <stdio.h>
2 # include <stdlib.h>
3 void main()
4 {
5 FILE* fpin,* fpout;
6 char ch,infile[10],outfile[10];
7 printf("请输入源文件名:\n");
8 scanf("%s",infile); /*输入源文件名*/
9 if((fpin=fopen(infile,"r"))==NULL) /*以读方式打开源文件*/
10 {
11 printf("can not open the source file.\n");
12 exit(0);
13 }
14 printf("请输入目标文件名:\n");
15 scanf("%s",outfile); /*输入目标文件名*/
16 if((fpout=fopen(outfile,"w"))==NULL) /*以写方式打开目标文件*/
17 {
18 printf("can not create the target file.\n");
19 exit(0);
20 }
21 while(! feof(fpin)) //feof 函数是检测是否到了文件尾
22 {
23 ch=fgetc(fpin);
24 fputc(ch,fpout);
25 }
26 fclose(fpin);
27 fclose(fpout);
28}
```

程序运行结果:

请输入源文件名:file1<回车>

请输入目标文件名:file2<回车>

如果 file1 文件已经存在,该程序将 file1 文件中的内容复制到 file2 文件。



## 2) 读写数据块函数

(1) 读数据块函数 fread( )。

fread 函数从 fp 所指定的文件向内存输入数据块。

函数原型为：

```
int fread (数据类型 *buffer, unsigned size, unsigned count, FILE *fp);
```

参数说明: buffer 是指针变量, 指向数据存放区域的首地址, size 为一次读入的字节数, count 为读取次数, fp 为输入文件指针。

返回值: 正常操作返回为 count 的值(不是字节数), 否则返回 0。

如果文件能正常打开, fread 函数可以读取任何类型的数据。例如:

```
float fx;
fread(fx, 4, 1, fp);
```

其中, fx 为实型数据变量, 该函数从 fp 所指定的文件读入四个字节, 送到 fx 变量中, 并且只读一次。

还可以用这个函数读取一个结构体类型的数据。例如:

```
struct student
{
 char Num[8];
 char Name[10];
 int Age;
}stu[30];
```

结构体数据有 30 个元素, 每个元素占 22 个字节, 如果学生数据存放在磁盘文件中, 可以用 for 语句依次读取这 30 个学生数据, 并存放在数组 stu 中。

```
for(k=0; k<30; k++)
 fread(&stu[k], sizeof(struct student), 1, fp);
```

想一想, 为什么要用 &stu[k]?

(2) 写数据块函数 fwrite( )。

fwrite 函数将 buffer 缓冲区中的数据输出到 fp 所指定的文件中去。

函数原型为:

```
int fwrite (数据类型 *buffer, unsigned size, unsigned count, FILE *fp);
```

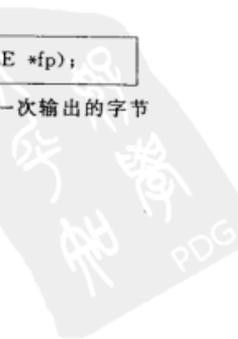
参数说明: buffer 是指针变量, 指向数据存放区域的首地址, size 为一次输出的字节数, count 为写入次数, fp 为输出文件指针。

返回值: 正常操作返回为 count 的值(不是字节数), 否则返回值为 0。

如果文件能正常打开, fwrite 函数可以写入任何类型的数据。

例如:

```
struct student stu[30];
for(k=0; k<30; k++)
```



```
fwrite(&stu[k],sizeof(struct student),1,fp);
```

该 for 语句将数组中 30 个元素依次写入 fp 所指定的文件中,并且只写一次。

**【例 11.3】** 从键盘上输入 3 个学生信息,然后将它们转存到磁盘文件中,并显示这些信息。

```
1 # include <stdio.h>
2 # include <stdlib.h>
3 # define N 3
4 void input(struct student stu[],int n);
5 void output(struct student stu[],int n);
6
7 struct student
8 {
9 char Num[7];
10 char Name[10];
11 int Age;
12 };
13
14 void main()
15 {
16 struct student stu[N];
17 input (stu,N);
18 output (stu,N);
19 }
20
21 void input(struct student stu[],int n)
22 {
23 FILE* fp;
24 int i;
25 if((fp=fopen("fbinary","wb"))==NULL) //以写方式开打二进制文件
26 {
27 printf("can not open this file.\n");
28 exit(0);
29 }
30 for(i=0;i<n;i++) //输入学生信息
31 {
32 printf("请输入学号:");
33 scanf("%s",stu[i].Num);
34 printf("请输入姓名:");
35 scanf("%s",stu[i].Name);
```

```
36 printf("请输入年龄:");
37 scanf("%d",&stu[i].Age);
38 }
39 for(i=0;i<n;i++) //保存学生信息
40 fwrite(&stu[i],sizeof(struct student),1,fp);
41 fclose(fp);
42 }
43
44 void output(struct student stu[],int n)
45 {
46 FILE* fp;
47 int i;
48 if((fp=fopen("fbinary","rb"))==NULL) //以读方式开打二进制文件
49 {
50 printf("can not open this file.\n");
51 exit(0);
52 }
53 for(i=0;i<n;i++)
54 {
55 fread(&stu[i],sizeof(struct student),1,fp);
56 printf("%7s%10s%5d\n",stu[i].Num,stu[i].Name,stu
 [i].Age);
57 }
58 fclose(fp);
59 }
```

程序运行结果:

请输入学号:1001<回车>

请输入姓名:张三<回车>

请输入年龄:19<回车>

请输入学号:1002<回车>

请输入姓名:李四<回车>

请输入年龄:18<回车>

请输入学号:1003<回车>

请输入姓名:王五<回车>

请输入年龄:21<回车>

|      |    |    |
|------|----|----|
| 1001 | 张三 | 19 |
| 1002 | 李四 | 18 |
| 1003 | 王五 | 20 |



### 11.2.3 文件的定位

前面介绍的文件读/写方式都是顺序读写,从文件的开头顺序读写每一个数据。实际上在一个文件的文件结构体中,都有一个指向当前读写位置的指针,在以读(或写)方式打开文件时,该指针指向文件的开头,如果以追加的方式打开文件,则该文件指针指向文件的末尾。在顺序读/写一个文件时,每次读/写都要修改文件指针的指向,使它指向下一次要读/写的位置。

我们也可以人为地控制当前文件指针的移动,可以让文件指针随意指向我们想要指向的位置,而不是像以往那样按物理顺序逐个移动,这就是所谓对文件的定位与随机读/写。

#### 1. 使指针指向文件开头的函数(`rewind()`函数)

按读/写方式打开文件时,文件指针指向文件开头,随着读/写文件的操作,文件指针的指向将发生变化。`rewind`函数可以在文件运行的过程中将文件指针重新移动到文件开头的位置。

函数原型为:

```
void rewind (FILE *fp);
```

函数功能:该函数使文件的读写位置指针重新指向文件的开头。

参数说明:fp 为文件指针。该函数没有返回值。

**【例 11.4】** `rewind` 函数使用举例。

```
1 # include <stdio.h>
2 # include <stdlib.h>
3 void main()
4 {
5 FILE* fp1,*fp2;
6 char ch;
7 if((fp1=fopen("file1","r"))==NULL)
8 {
9 printf("can not open the file.\n");
10 exit(0);
11 }
12 if((fp2=fopen("file2","w"))==NULL)
13 {
14 printf("can not open the file.\n");
15 exit(0);
16 }
17 while(!feof(fp1))
18 {
```

```

19 ch=getc(fp1);
20 putc(ch,fp2);
21 }
22 rewind(fp1); //重新将文件指针定位到文件头
23 while(!feof(fp1))
24 {
25 ch=getc(fp1);
26 putchar(ch);
27 }
28 putchar('\n');
29 fclose(fp1);
30 fclose(fp2);
31 }

```

程序运行结果:

```
One World One Deram
```

和前面一样,该程序运行前在当前目录下必须要有 file1 文件,通过程序的运行将 file1 文件拷贝到 file2 文件中,并且在屏幕上输出 file1 文件内容。

## 2. 设置文件指针位置函数(fseek()函数)

函数原型为:

```
int fseek(FILE *fp,long offset,int fromwhere);
```

函数功能:把 fp 所指定文件的读写位置指针设置到相对 fromwhere 位移量为 offset 的地方。

参数说明:fp 为文件指针,offset 为相对位移量(即相对于 fromwhere 的位移量),可以为正数也可以为负数,如果为正数,指针向地址高的方向移动,如果为负数,指针向地址低的方向移动,fromwhere 是相对位移的基点,必须是 0、1、2 中的一个,分别代表以下三个符号常量:

| fromwhere | 命名       | 含义       |
|-----------|----------|----------|
| 0         | SEEK_SET | 文件开始     |
| 1         | SEEK_CUR | 当前文件指针位置 |
| 2         | SEEK_END | 文件末尾     |

返回值:如果指针成功地移动了,返回值为零,否则返回一个非零值。

请看下面例子:

```
fseek(fp,5L,SEEK_SET);
```

将文件指针移动到离文件头 5 个字节的位置。

fseek(fp,20L,1); 将位置指针移到离当前位置 20 个字节的位置(向地址高的方向移动)。

fseek(fp,-20L,1);将位置指针移到离当前位置 20 个字节的位置(向地址低的方向

移动)。

fseek(fp, -30L, 2);将位置指针移到距离文件末尾 30 个字节的位置。

从上面的例子可以看出:

① 使用 SEEK\_SET、SEEK\_CUR、SEEK\_END 与使用 0、1、2 是等价的;

② 如果使用 SEEK\_SET, 文件指针只能向地址高的方向移动, 即 offset 只能使用正数; 如果用 SEEK\_CUR, 文件指针既可以向地址高的方向移动也能向地址低的方向移动, offset 既能用正数, 也能用负数; 如果使用 SEEK\_END, 文件指针只能向地址低的方向移动, 即 offset 只能用负数。

**【例 11.5】** 输入 5 个学生的数据并保存在磁盘文件上, 在屏幕上显示第 1、3、5 个学生信息。

```

1 # include <stdio.h>
2 # include <stdlib.h>
3 # define SIZE 5
4 struct student
5 {
6 int num;
7 char name[10];
8 int age;
9 char sex[3];
10 }stud[SIZE];
11
12 void save ()
13 {
14 int i;
15 FILE* fp;
16 if ((fp=fopen("stud_dat", "wb"))==NULL)
17 {
18 printf("can not open file\n");
19 exit(0);
20 }
21 for(i=0;i<SIZE;i++)
22 if(fwrite(&stud[i], sizeof(struct student), 1, fp)!=1)
23 printf("file write error\n");
24 fclose(fp);
25 }
26
27 void output ()
28 {
29 int i;

```



```
30 FILE* fp;
31 if((fp=fopen("stud_dat","rb"))==NULL)
32 {
33 printf("can not open file\n");
34 exit(0);
35 }
36 printf("输出学生信息:\n");
37 for(i=0;i<SIZE;i+=2)
38 {
39 fseek(fp,i*sizeof(struct student),0);
40 fread(&stud[i],sizeof(struct student),1,fp);
41 printf("%7d%10s%4d%4s\n",stud[i].num,stud[i].name,
42 stud[i].age,stud[i].sex);
43 }
44 fclose(fp);
45 }
46 void main()
47 {
48 int i;
49 printf("请输入学生学号、姓名、年龄、性别(用空格分隔)\n");
50 for(i=0;i<SIZE;i++)
51 scanf("%d%s%d%s",&stud[i].num,stud[i].name,&stud[i].
52 age,stud[i].sex);
53 save();
54 output();
55 }
```

程序运行结果:

请输入学生学号、姓名、年龄、性别(用空格分隔)

1001 张三 21 男  
1002 李四 22 女  
1003 王五 23 男  
1004 马六 19 女  
1005 钱七 18 男

输出学生信息:

|      |    |    |   |
|------|----|----|---|
| 1001 | 张三 | 21 | 男 |
| 1003 | 王五 | 23 | 男 |
| 1005 | 钱七 | 18 | 男 |



### 3. 获得文件当前读/写位置函数(ftell()函数)

函数原型为:

```
long ftell(FILE * fp);
```

函数功能:该函数用于取得流式文件当前的读/写位置,它是用相对于文件开头位移量来表示该位置的。

参数说明:fp 为文件指针。

返回值:如果正常,返回值为位移量,否则返回值为-1。

例如:

```
if(ftell(fp)==- 1)
 printf("\a Error!\n");
```

字符'\a'为转义字符,在这里表示计算机扬声器响铃一次,以示报警。

## 11.3 扩展知识与理论

### 11.3.1 读/写字符串和格式化读/写数据函数

#### 1. 读/写字符串函数

##### 1) 读字符串函数 fgets( )

fgets 的作用是从指定文件向内存输入一个字符串。

函数原型为:

```
char * fgets(char * str,int n,FILE * fp);
```

参数说明:str 为读取到的字符串的地址,可以是指针,也可以是数组,n 为限定每次读取的字符个数,fp 为指定读取的文件指针。

返回值:正常操作返回值为 str 的首地址,当读到文件末尾或出错时,返回 NULL。

注意:从 fp 所指向的文件当前读写位置开始,最多读入 n-1 个字符,同时将字符串结束标志'\0'也复制到字符数组 str 中,从文件中读入一个字符串后,读写位置将后移到该字符串的下一个字符处。

用 fgets( )函数读取字符串时,遇到下列情况中任何一种,读取过程将结束:

已读取了 n-1 个字符;

读到换行符;

读到文件末尾。

##### 2) 写字符串函数 fputs( )

fputs 的作用是从内存向指定文件输出一个字符串。

函数原型为：

```
int fputs(char * str, FILE * fp);
```

参数说明: str 为指定输出的字符串,它可以是指针、数组名或字符串,fp 为指定的输出文件。

返回值:正常操作返回值为所输出的字符串中最后一个字符的 ASCII 值,如果向文件写入字符串不成功,则返回值为 EOF。

在使用 fputs 函数时,有一点是需要加以注意的:fputs(str,fp)将舍去字符串结束标志'\0',而 puts(str)则将字符串结束标志'\0'转换为回车符输出。这就要求我们在使用 fputs 函数时,磁盘文件向 str 传递完后,应再加上一个字符串结束标志,一般的格式为:

```
fputs(str,fp);
fputc('\n',fp);
```

**【例 11.6】** fgets 与 fputs 函数举例。

```
1 # include <stdio.h>
2 # include <stdlib.h>
3 void disp_filec(FILE* fp); //函数声明
4 void disp_files(FILE* fp, int n);
5 void main()
6 {
7 FILE* fp;
8 char fname[81];
9 char s[81];
10 printf("请输入文件名:");
11 scanf("%s", fname);
12 if((fp=fopen("fname", "w+"))==NULL)
13 {
14 printf("can not open this file.\n");
15 }
16 while(gets(s)!=NULL)
17 {
18 fputs(s,fp);
19 fputc('\n',fp); //加上回车换行符
20 }
21 putchar('\n');
22 rewind(fp); //重新将文件指针指向文件头
23 disp_filec(fp);
24 rewind(fp); //重新将文件指针指向文件头
25 disp_files(fp,81);
26 fclose(fp);
```



```

27 }
28
29 void disp_filec(FILE* fp)
30 {
31 char ch;
32 while((ch=fgetc(fp))!=EOF)
33 putchar(ch);
34 }
35
36 void disp_files(FILE* fp,int n)
37 {
38 char str[81];
39 while(fgets(str,n,fp)!=NULL)
40 puts(str);
41 }

```

程序运行结果:

```

请输入文件名:file3
湖北恩施土家族苗族自治州<回车>
恩施职业技术学院计算机系<回车>
^Z<回车>

```

```

湖北恩施土家族苗族自治州
恩施职业技术学院计算机系

湖北恩施土家族苗族自治州

恩施职业技术学院计算机系

```

请注意输出格式的不同(字符串输出多一空行),想一想这是为什么?

## 2. 格式化读/写函数

以前学习的格式化输入/输出是针对终端的,即 scanf 函数只针对键盘,而 printf 函数只针对显示器,如果输入/输出对象是磁盘文件,可以采用 fscanf 函数和 fprintf 函数。

### 1) 格式化读函数 fscanf( )

fscanf 函数按照“输入格式字符”所指定的输入格式,从 fp 指定文件的当前位置向内存储输入数据,然后把它们按输入项地址表的顺序存入指定的存储单元中。

函数原型为:

```
int fscanf(FILE *fp,“输入格式字符串”,输入项地址表);
```

参数说明:fp 为指定的输入文件;“输入格式字符串”与 scanf 函数中的输入格式字符串相同;输入项地址表为从指定文件中读入数据的存放地址,如果有多个输入项,输入项之间用逗号隔开。

返回值:如果操作成功,返回值为非零值,即输入的数据的个数,如果失败,返回零值,如果读到文件尾,返回 EOF。

输入数据后,fp 指针读写位置将移动到输入数据之后。

## 2) 格式化写函数 fprintf( )

fprintf 函数把内存中的数据按输出项表格式输出到 fp 所指定的文件中。

函数原型为:

```
int fprintf(FILE *fp,“输出格式字符串”,输出项表);
```

参数说明:fp 是文件指针,它指向输出文件;“输出格式字符串”为给定的输出格式,与 printf 函数的输出格式字符串相同;输出项表为输出对象,如果有多个输出项,输出项之间用逗号隔开。

返回值:如果正常操作,返回值为非零值,否则为 EOF 值。

向文件输出数据后,文件的读写位置将移动到所写入的数据之后。

**【例 11.7】** 将例 11.3 改用格式化读写函数实现其功能。

用格式化读写函数去替代读写数据块函数非常简单,只要将例 11.3 第 40 行改为:

```
fprintf(fp, "%7s%10s%5d", stu[i].Num, stu[i].Name, stu[i].Age);
```

第 55 行改为:

```
fscanf(fp, "%7s%10s%5d", stu[i].Num, stu[i].Name, &stu[i].Age);
```

其他不变即可。

程序运行结果与例 11.3 相同。

**【例 11.8】** 现设学生信息包括学号、姓名、三门课的单科成绩和平均分,创建一个名为 score.txt 的文本文件,从键盘输入数据,当学号为 0 时退出输入,用格式化读写数据到磁盘文件,然后显示该数据到屏幕。

```
1 # include <stdio.h>
2 # include <stdlib.h>
3 # define N 30 //定义最多能处理学生数
4 void input(struct student* sp);
5 void output(struct student* sp);
6
7 struct student
8 {
9 char Num[5];
10 char Name[10];
11 float score[3]; //三门功课成绩
12 float ave; //平均成绩
```



```
13 };
14
15 void main ()
16 {
17 struct student stu[N]; //定义学生数组
18 input (stu); //调用输入函数
19 output (stu); //调用输出函数
20 }
21
22 void input (struct student * sp)
23 {
24 FILE * fp;
25 int i;
26 if ((fp=fopen("stuscore.txt", "w"))==NULL)
27 {
28 printf("can not open the file.\n");
29 exit(0);
30 }
31 printf("请输入学号(学号为0结束输入):");
32 scanf("%5s", sp->Num);
33 while(atoi(sp->Num)!=0) //atoi()将字符串转变成整型数据
34 {
35 printf("请输入姓名:");
36 scanf("%10s", sp->Name);
37 sp->ave=0;
38 for(i=0; i<3; i++)
39 {
40 printf("请输入第%d门功课成绩:", i+1);
41 scanf("%f", &sp->score[i]);
42 sp->ave+=sp->score[i];
43 }
44 sp->ave/=3.0f;
45 fprintf(fp, "%5s%10s", sp->Num, sp->Name);
46 for(i=0; i<3; i++)
47 fprintf(fp, "%10.1f", sp->score[i]);
48 fprintf(fp, "%10.1f", sp->ave);
49 sp++;
50 printf("请输入学号(学号为0结束输入):");
51 scanf("%5s", sp->Num);
```



```
52 }
53 fclose(fp);
54 }
55
56 void output(struct student* sp)
57 {
58 FILE* fp;
59 if((fp=fopen("stuscore.txt","r"))==NULL)
60 {
61 printf("can not open the file.\n");
62 exit(0);
63 }
64 printf(" 学号 姓名 成绩 1 成绩 2 成绩 3
 平均成绩\n");
65 while(fscanf(fp,"%s%s%f%f%f",sp->Num,sp->Name,&sp->
 score[0],&sp->score[1],&sp->score[2],&sp->ave)!
 =EOF)
66 {
67 printf("%5s%10s%10.1f%10.1f%10.1f%10.1f\n",sp->
 Num,sp->Name,sp->score[0],sp->score[1],sp
 ->score[2],sp->ave);
68 sp++;
69 }
70 fclose(fp);
71 }
```

程序运行结果:

```
请输入学号(学号为0结束输入):1001<回车>
请输入姓名:张三<回车>
请输入第1门功课成绩:98<回车>
请输入第2门功课成绩:97<回车>
请输入第3门功课成绩:95<回车>
请输入学号(学号为0结束输入):1002<回车>
请输入姓名:李四<回车>
请输入第1门功课成绩:90<回车>
请输入第2门功课成绩:98<回车>
请输入第3门功课成绩:87<回车>
请输入学号(学号为0结束输入):0<回车>
```



| 学号   | 姓名 | 成绩 1 | 成绩 2 | 成绩 3 | 平均成绩 |
|------|----|------|------|------|------|
| 1001 | 张三 | 98.0 | 97.0 | 95.0 | 95.0 |
| 1002 | 李四 | 90.0 | 98.0 | 87.0 | 91.7 |

### 11.3.2 文件状态检测

在进行文件读/写时,程序员对文件状态要做到心中有数,这样才能对文件进行正确读/写。C语言提供了一些函数来检测文件的状态和进行文件出错提示,以帮助程序员了解文件读/写情况。下面对这些函数进行介绍。

#### 1. 文件读/写结束标志函数(feof( )函数)

函数原型为:

```
int feof(FILE *fp);
```

函数功能:用来检查文件读/写是否结束。

参数说明:fp 是文件指针。

返回值:如果文件正常结束,返回值为 1,如果文件没有结束返回值为 0。

**【例 11.9】** 通过命令行来传递文件名,最终完成两个文件的复制。

```
1 # include <stdio.h>
2 # include <stdlib.h>
3 # define SIZE 10
4 struct student
5 {
6 int num;
7 char name[10];
8 int age;
9 char sex[3];
10 }stud[SIZE];
11 void main(int argc,char*argv[])
12 {
13 FILE*fp1,*fp2;
14 int i=0;
15 if(argc!=3)
16 {
17 printf("Error in command!\n");
18 exit(0);
19 }
20 if((fp1=fopen(argv[1],"rb"))==NULL)
21 {
```



```
22 printf("can not open this file\n");
23 exit(0);
24 }
25 if((fp2=fopen(argv[2], "wb"))==NULL)
26 {
27 printf("can not open this file\n");
28 exit(0);
29 }
30 while(!feof(fp1))
31 {
32 fread(&stud[i], sizeof(struct student), 1, fp1);
33 fwrite(&stud[i], sizeof(struct student), 1, fp2);
34 i++;
35 }
36 fclose(fp1);
37 fclose(fp2);
38 }
```

程序运行结果:

```
E:\> e11_9 stud_dat stud2<回车>
```

268 该程序要在 DOS 环境中运行,上面的 E:\> 是 DOS 提示符, e11\_9 是例 11.9 开发成功后的可执行文件名, stud\_dat 是已有文件, stud2 是执行 e11\_9 文件后所复制的文件。执行完后可以在与 e11\_9 和 stud\_dat 所在的相同文件夹中找到 stud2 文件,它与 stud\_dat 文件相同。

## 2. 检查文件出错函数(ferror( )函数)

函数原型为:

```
int ferror(FILE *fp);
```

函数功能:在调用各种输入/输出函数时,可用该函数检查是否出错。

参数说明:fp 为文件指针。

返回值:如果文件调用没有出错,返回值为 0,否则返回值为非 0 值。

由于该函数是用来检查输入/输出函数的每次调用是否出错,因此应该在调用输入/输出函数后立即调用此函数,以检查输入/输出函数的引用是否正确。

例如:

```
ch=fgetc(fp);
if(ferror(fp))printf("Error in I/O.\n");
```

### 11.3.3 常见错误及处理方法

常见错误 1:使用文件之前忘记打开文件,使用完文件后忘记关闭文件。

常见错误 2: 调用打开文件操作后不进行相应的检查。

用 r、rb 方式打开一个并不存在的文件时, 如果不进行打开文件操作检查, 将不能进行相应的操作, 所以对文件打开操作的情况要进行相应的检查。

常见错误 3: 用只读方式打开, 却企图向该文件写数据。

```
if (fp = fopen("test", "r")) == NULL)
{
 printf("cannot open this file\n");
 exit(0);
}
ch = fgetc(fp);
while (ch != '#')
{
 ch = ch + 4;
 fputc(ch, fp); // 错误
 ch = fgetc(fp);
}
```

## 11.4 深入训练

269

- (1) 从键盘上输入 10 个整数, 分别以文本文件和二进制文件方式存入磁盘。
- (2) 编程实现一个文本文件的复制。
- (3) 有 3 个学生, 每个学生有 3 门课的成绩, 从键盘上分别输入每个学生的学号、姓名和 3 门课的成绩, 保存到一个名为 class31.dat 的文本文件中去。
- (4) 有 3 个学生, 每个学生有 3 门课的成绩, 从键盘上分别输入每个学生的学号、姓名和 3 门课的成绩, 保存到一个名为 class32.dat 的二进制文件中去。
- (5) 打开上题建立的二进制文件, 按“学号、姓名、成绩 1、成绩 2、成绩 3、平均成绩”的格式把文件中的记录全部显示出来, 并且要求把学生的总人数在屏幕上显示出来。

## 习 题 11

### 11.1 填空题

- (1) 在 C 语言中, 文件可以用\_\_\_\_\_方式存取, 也可以用\_\_\_\_\_存取。
- (2) 打开文件的含义是\_\_\_\_\_, 关闭文件的含义是\_\_\_\_\_。
- (3) 文本文件在内存中以\_\_\_\_\_方式存储, 二进制文件在内存中以\_\_\_\_\_方式存储。
- (4) fopen() 函数有两个形式参数, 一个表示\_\_\_\_\_, 另一个表示\_\_\_\_\_。
- (5) EOF 可以用来判断文本文件是否结束, 如果遇到文件结束, EOF 值为

\_\_\_\_\_，否则为\_\_\_\_\_。

(6) feof(fp) 函数用来判断文件是否结束，如果遇到文件结束，函数值为\_\_\_\_\_，否则为\_\_\_\_\_。

(7) 下面程序由终端键盘输入字符，存放于文件中，用“!”结束输入。请在横线处填入适当内容。

```
#include <stdio.h>
#include <stdlib.h>
main ()
{
 FILE* fp;
 char ch, fname[10];
 printf("Input name of file\n");
 gets(fname);
 if ((fp=fopen(fname, "w"))==NULL)
 {
 printf("cannot open\n");
 exit(0);
 }
 printf("Enter data:\n");
 while(_____'!' != '!')
 fputc(_____) ;
 fclose(fp);
}
```

(8) 下面程序从一个二进制文件中读入结构体数据，并把结构体数据显示在终端屏幕上。请在横线处填入适当内容。

```
include <stdio.h>
struct rec
{
 int num;
 float total;
};
main()
{
 FILE* f;
 f=fopen("bin.dat", "rb");
 reout(f);
 fclose(f);
}
reout(_____)
```



```

{
 struct rec r;
 while(!feof(f))
 {
 fread(&r, _____, 1, f);
 printf("%d, %f\n", _____);
 }
}

```

### 11.2 判断题(判断下列叙述的正确性,正确的请打“√”,错误的请打“×”)

- (1) 文件操作前必须先打开,操作结束后必须关闭。 ( )
- (2) 打开文件就是建立文件指针与文件的联系,关闭文件就是将这种联系切断。 ( )
- (3) 如果用 fopen 函数不能正常打开文件,函数的返回值为 0。 ( )
- (4) 文件正常操作时,文件指针始终不会移动。 ( )
- (5) 用 fread 函数能够指定每次读取的字节数和读取次数。 ( )
- (6) 用 fgets 函数读取字符串时,不会读取字符串结束标志'\0'。 ( )

### 11.3 选择题

(1) 系统的标准输入文件是( )。

- A. 键盘            B. 显示器            C. 软盘            D. 硬盘

(2) 以下可作为函数 fopen 中第一个参数的正确格式是( )。

- A. c:\user\text.txt            B. c:\user\text.txt  
C. "c:\user\text.txt"            D. "c:\\user\\text.txt"

(3) 若要用 fopen 函数打开一个新的二进制文件,该文件要既能读也能写,则文件打开方式字符串应是( )。

- A. "ab+"            B. "wb+"            C. "rb+"            D. "ab"

(4) fgetc 函数的作用是从指定的文件读入一个字符,该文件的打开方式必须是( )。

- A. 只写            B. 追加            C. 读或读写            D. B 和 C 都正确

(5) fp 是指向某文件的指针,且已读到文件的末尾,则表达式 feof(fp)正确的返回值是( )。

- A. NULL            B. 0            C. 非零值            D. EOF

(6) fscanf( )函数的正确调用形式是( )。

- A. fscanf(格式字符串,输入项地址表,fp);  
B. fscanf(fp,格式字符串,输入项地址表);  
C. fscanf(格式字符串,文件指针,输入项地址表);  
D. fscanf(文件指针,格式字符串,输出项表);

(7) 已知函数的调用形式: fread(buffer, size, count, fp); 其中 buffer 代表的是( )。

- A. 一个整型变量,代表要读入的数据项总数

- B. 一个文件指针,指向要读的文件
- C. 一个指针,指向要读入的数据在存放地址
- D. 一个存储区,存放要读的数据项

(8) 设有以下结构体类型:

```
struct student
{
 char name[8];
 int num;
 float score[4];
}stu [50];
```

假设结构体数组中的元素都已有值,若要将这些元素写到硬盘文件 fp 中,以下不正确的形式是( )。

- A. fwrite(stu,sizeof(struct student),50,fp);
- B. fwrite(stu,50\*sizeof(struct student),1,fp);
- C. fwrite(stu,25\*sizeof(struct student),25,fp);
- D. for(i=0;i<50;i++)  
    fwrite(stu+i,sizeof(struct student),1,fp);

(9) 阅读下列程序,程序功能的正确描述是( )。

```
include <stdio.h>
include <stdlib.h>
void main()
{
 FILE* in,*out;
 char infile[10],outfile[10];
 printf("Enter the infile name:\n");
 scanf("%s",infile);
 printf("Enter the outfile name:\n");
 scanf("%s",outfile);
 if((in=fopen(infile,"r"))==NULL)
 {
 printf("cannot open infile\n");
 exit(0);
 }
 if((out=fopen(outfile,"w"))==NULL)
 {
 printf("cannot open outfile\n");
 exit(0);
 }
 while(!feof(in))
```



```
 fputc(fgetc(in),out);
 fclose(in);
 fclose(out);
}
```

- A. 程序完成将磁盘文件的信息在屏幕上显示的功能
- B. 程序完成将两个磁盘文件合二为一的功能
- C. 程序完成将一个磁盘文件复制到另一个磁盘文件中
- D. 程序完成将两个磁盘文件合并并且在屏幕上输出



# 附 录

## 一、常用字符与 ASCII 码对照表

| ASCII 码 | 字符  | ASCII 码 | 字符    | ASCII 码 | 字符 | ASCII 码 | 字符 | ASCII 码 | 字符 | ASCII 码 | 字符  |
|---------|-----|---------|-------|---------|----|---------|----|---------|----|---------|-----|
| 000     | NUL | 022     | SYN   | 044     | ,  | 066     | B  | 088     | X  | 110     | n   |
| 001     | SOH | 023     | ETB   | 045     | -  | 067     | C  | 089     | Y  | 111     | o   |
| 002     | STX | 024     | CAN   | 046     | .  | 068     | D  | 090     | Z  | 112     | p   |
| 003     | ETX | 025     | EM    | 047     | /  | 069     | E  | 091     | [  | 113     | q   |
| 004     | EOT | 026     | SUB   | 048     | 0  | 070     | F  | 092     | \  | 114     | r   |
| 005     | EDQ | 027     | ESC   | 049     | 1  | 071     | G  | 093     | ]  | 115     | s   |
| 006     | ACK | 028     | FS    | 050     | 2  | 072     | H  | 094     | -  | 116     | t   |
| 007     | BEL | 029     | GS    | 051     | 3  | 073     | I  | 095     | _  | 117     | u   |
| 008     | BS  | 030     | RS    | 052     | 4  | 074     | J  | 096     | '  | 118     | v   |
| 009     | HT  | 031     | US    | 053     | 5  | 075     | K  | 097     | a  | 119     | w   |
| 010     | LF  | 032     | space | 054     | 6  | 076     | L  | 098     | b  | 120     | x   |
| 011     | VT  | 033     | !     | 055     | 7  | 077     | M  | 099     | c  | 121     | y   |
| 012     | FF  | 034     | "     | 056     | 8  | 078     | N  | 100     | d  | 122     | z   |
| 013     | CR  | 035     | #     | 057     | 9  | 079     | O  | 101     | e  | 123     | {   |
| 014     | SO  | 036     | \$    | 058     | :  | 080     | P  | 102     | f  | 124     |     |
| 015     | SI  | 037     | %     | 059     | ;  | 081     | Q  | 103     | g  | 125     | }   |
| 016     | DLE | 038     | &     | 060     | <  | 082     | R  | 104     | h  | 126     | ~   |
| 017     | DC1 | 039     | '     | 061     | =  | 083     | S  | 105     | i  | 127     | DEL |
| 018     | DC2 | 040     | (     | 062     | >  | 084     | T  | 106     | j  |         |     |
| 019     | DC3 | 041     | )     | 063     | ?  | 085     | U  | 107     | k  |         |     |
| 020     | DC4 | 042     | *     | 064     | @  | 086     | V  | 108     | l  |         |     |
| 021     | NAK | 043     | +     | 065     | A  | 087     | W  | 109     | m  |         |     |

## 二、32 个关键字

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

## 三、9 种控制语句

- |                 |                     |
|-----------------|---------------------|
| (1) if( )~else~ | (条件语句)              |
| (2) for( ) ~    | (循环语句)              |
| (3) while( ) ~  | (循环语句)              |
| (4) do-while( ) | (循环语句)              |
| (5) continue    | (结束本次循环语句)          |
| (6) break       | (中止执行 switch 或循环语句) |
| (7) switch      | (多分支选择语句)           |
| (8) goto        | (转向语句)              |
| (9) return      | (函数返回语句)            |

275

## 四、运算符的优先级和结合性

| 优先级 | 运算符                           | 结合性  |
|-----|-------------------------------|------|
| 15  | () [] -> .                    | 自左至右 |
| 14  | ! ~ + - ++ -- & * (类型) sizeof | 自右至左 |
| 13  | * / %                         | 自左至右 |
| 12  | + -                           |      |
| 11  | << >>                         |      |
| 10  | < <= > >=                     |      |

续表

| 优先级 | 运算符                               | 结合性  |
|-----|-----------------------------------|------|
| 9   | == !=                             | 自左至右 |
| 8   | &                                 |      |
| 7   | ^                                 |      |
| 6   |                                   |      |
| 5   | &&                                |      |
| 4   |                                   |      |
| 3   | ? :                               |      |
| 2   | = *= /= %= += -= &= ^=  = <<= >>= | 自右至左 |
| 1   | ,                                 | 自左至右 |

## 五、常用标准函数及其头文件

### 1. 字符串处理函数, 头文件 string. h

char \* strcpy(char \* destin, char \* source);

复制字符串将 source 复制到 destin 中。

char \* strcat(char \* destin, char \* source);

连接字符串将 source 连接到 destin 字符串的后面。

int strcmp(char \* str1, char \* str2);

比较两字符串, str1 大于 str2 时返回正数, str1 小于 str2 时返回负数, str1 等于 str2 时返回 0。

int strlen(char \* str);

返回字符串 str 的长度。

char \* strchr(char \* str, char c);

返回 s 中第一个等于字符 c 的字符地址, 若不存在返回 NULL。

char \* strstr(char \* str1, char str2);

在 str1 中查找子串 str2, 若找到, 返回其起始地址, 否则返回 NULL。

char \* strdup(char \* str);

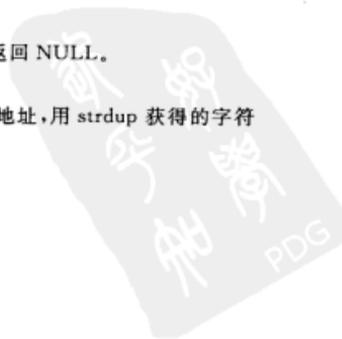
将 str 复制到用 malloc 申请的动态空间中并返回该空间地址, 用 strdup 获得的字符串副本必须用 free 函数释放。

### 2. 数学函数, 头文件 math. h

int abs(int d);

返回 d 的绝对值。

double fabs(double d);



返回 d 的绝对值。

```
long labs(long d);
```

返回 d 的绝对值。

```
double sin(double d);
```

返回 d 的正弦。

```
double cos(double d);
```

返回 d 的余弦。

```
double pow(double a,double x);
```

返回 a 的 x 次方。

```
double sqrt(double d);
```

返回 d 的平方根。

```
double log(double d);
```

返回 d 的自然对数值。

```
double log10(double d);
```

返回 d 的常用对数值。

### 3. 其他函数,头文件 `stdlib.h`

本头文件中的函数较杂,大体可以分为动态内存管理函数、数学转换函数等,下面分别列举几个。

#### (1) 动态内存管理函数

```
void * malloc(unsigned size);
```

申请一块 size 大小的动态空间,返回该空间的首地址。

```
void free(void * p);
```

释放 p 所指向的用 malloc 函数申请的空间。

#### (2) 随机数函数

```
int rand(void);
```

返回一个 0 到 `RAND_MAX`(`stdlib.h` 中定义的符号常量)之间的伪随机数。

```
void srand(unsigned seed);
```

用种子 seed 重新初始化随机数生成器。

```
int random(int unum);
```

返回一个从 0 到 `(num-1)` 的随机数。

#### (3) 数学转换函数

```
double atof(char * nptr);
```

将 nptr 所指的字符串转换成 double 类型。

```
int atoi(char * nptr);
```

将 nptr 所指的字符串转换成 int 类型。

```
long atoll(char nptr);
```

将 nptr 所指的字符串转换成 long 类型。

```
char * itoa(int value char * s,int base);
```



将整数 value 按数制 base 转换成字符串存入 s 中并返回指针 s。

```
char * ltoa(long value, char * s, int base);
```

将长整数 value 按数制 base 转换成字符串存入 s 中并返回指针 s。

```
char * ultoa(unsigned long value, char * s, int base);
```

将无符号长整数 value 按数制 base 转换成字符串存入 s 中并返回指针 s。

#### (4) 终止程序函数

```
void exit(int status);
```

exit 终止调用进程,在退出以前,所有文件被关闭,缓冲输出(正等待输出)内容被写完。

#### (5) 调用 DOS 命令函数

```
int system(char * command);
```

system 发出一个 MS-DOS 命令。通过 MS-DOS 的 command.com 文件执行由字符串 command 给定的命令,就像在 DOS 提示符下敲入该命令一样。

### 4. 控制台函数,头文件 conio.h

```
int getch(void);
```

getch 从控制台无回显地取一字符。



## 参 考 文 献

- [1] 谭浩强. C 语言程序设计[M]. 北京:清华大学出版社,1999.
- [2] 李宁. C++ 语言程序设计[M]. 北京:中央广播电视大学出版社,2004.
- [3] 谭浩强. C 程序设计试题汇编[M]. 北京:清华大学出版社,1998.
- [4] 王建平. C 语言程序设计实验指导与习题详解[M]. 北京:中国水利水电出版社,2001.

