C 语言程序设计项目化教程

主　编　付　琳　梁英坚

西北工业大学出版社

西　安

目　　录

项目一　实用计算器

项目设置目的

本项目以实用计算器为背景，介绍C语言的数据类型和运算、程序的控制结构等。按照理论知识的先后顺序，将项目分解为三个任务。通过理论知识的学习和项目的实现，学生可了解C语言编程特点并可以编写简单的程序。

项目分析

本项目要求所编写、设计的计算器能够完成基本的数学运算功能。在设计时先从整体出发，再考虑局部细节，也就是“自顶向下，逐步细化”的设计原则。根据这一原则，分析的思路是：计算器功能→组成计算器的主要功能→实现每个功能。

实用计算器功能为实现整型数据和实型数据的加、减、乘、除四则运算。为增加用户体验，首先提供系统操作界面，给出加、减、乘、除和退出5个选项。在用户选择某一选项后（退出选项除外），系统提示输入第一个运算数和第二个运算数，然后系统自动计算出对应的四种运算结果。最后询问用户是否继续，如果输入字母x或y，重新返回主菜单；如果输入其他字母，则结束计算并退出系统。另外，为了使用方便，主菜单中特设0选项，选择该选项也能正常退出系统。

项目分解

任务一　认识C语言程序；
任务二　实用计算器中的数据类型与运算；
任务三　实用计算器中菜单的设计。

任务一　认识C语言程序

学习目标

- 掌握Visual C++ 6.0（简称“VC++ 6.0”）的C语言程序开发流程；
- 熟练掌握C语言的基本数据类型、常量和变量、运算符和表达式；
- 理解输入/输出函数、顺序结构程序设计、选择结构程序设计和循环结构程序设计；
- 初步掌握利用C语言进行软件开发的基本方法和步骤。

一、任务描述

实现实用计算器中的系统操作界面。

二、知识要点

本任务主要涉及 VC++6.0 的使用，新建项目工程和新建、保存文件等。

三、任务分析

根据系统的功能设计出系统操作菜单界面。

（1）新建工程和文件。

（2）使用 C 语言提供的标准输出函数 printf（）打印出界面，“\t”代表水平制表（一个 Tab 位置），“\n”代表换行。读者也可自行设计界面显示样式，设计应做到美观、友好。

四、源代码参考

```
#include <stdio.h>
void main()
{
    printf("\n\n");
    printf("\t\t|**************************************************************|\n");
    printf("\t\t|                        实用计算器                            |\n");
    printf("\t\t  |----------------------|\n");
    printf("\t\t|                          1—加法                              |\n");
    printf("\t\t|                          2—减法                              |\n");
    printf("\t\t|                          3—乘法                              |\n");
    printf("\t\t|                          4—除法                              |\n");
    printf("\t\t|                          0—退出                              |\n");
    printf("\t\t|**************************************************************|\n");
    printf("\n\n");
}
```

1.1 C 语言的特点及发展

C 语言是一门通用计算机编程语言，应用广泛。它设计精巧、功能齐全，既适合于编写应用软件，又适合于编写系统软件。

1.1.1 C 语言的特点

C 语言以其简捷、灵活、表达能力强、产生的目标代码质量高、可移植性强等特点而著称于世。一种语言要具有长久生命力，总需有不同于其他语言的特点。归纳起来，C 语言具有下列特点。

1. C 语言是中级语言

C 语言是处于汇编语言和高级语言之间的一种中间型程序设计语言，常被称为中级语言。C 语言把高级语言的基本结构和汇编语言的高效率结合起来。一方面，C 语言具有高级语言面向用户、容易理解、便于阅读和书写的优点；另一方面，C 语言可以和汇编语言一样对位、字节和地址进行操作，具有直接访问硬件的功能。

2. C 语言是一种结构化语言

C 语言具有编写结构化程序所必需的基本流程控制语句。C 语言程序是由函数集合构成的，各函数除了必要的信息交流外彼此独立，可方便地调用。这种结构化方式可使程序层次清晰，便于使用、维护以及调试。

3. C 语言功能齐全

C 语言具有丰富的运算符，表达式类型多样化，可以实现其他高级语言难以实现的运算。C 语言数据类型丰富，尤其是引入了指针概念，使得程序效率更高。

4. C 语言简捷、紧凑，使用方便、灵活

C 语言的语法限制不太严格，编程自由度大，例如对数组的下标出界、函数参数虚实转换不做检查。变量类型的使用也比较灵活，例如整型、字符型和逻辑型数据在特定条件下可以通用等，这为编程带来了极大方便。

5. C 语言可移植性强

C 语言程序本身并不依赖于计算机硬件系统和操作系统，因此在不同硬件结构和运行不同操作系统的计算机间基本不作修改就可以实现程序的移植。C 语言提供预处理命令，可以提高软件开发效率，并为程序的组织和编译提供了便利，也提高了程序的可移植性。

C 语言是众多后继课程的基本编程工具，特别是与 Windows 编程有关的课程。因此，与计算机相关的专业把 C 语言程序设计列为基础课程之一。学好 C 语言，对将来学习其他程序设计相关课程具有重要意义。

1.1.2 C 语言的发展

C 语言是由 B 语言发展而来的，而 B 语言又是由 A 语言发展而来的。

A 语言是指高级语言 ALGOL 60。1960 年出现的 ALGOL 60 是一种面向问题的过程式高级语言。以前的操作系统等系统软件主要是用汇编语言编写的，但汇编语言的可读性和可移植性比较差，因此想改用高级语言。而 ALGOL 60 离硬件较远，不适宜用来编写系统软件。

1963 年英国剑桥大学推出了 CPL（Combined Programming Language）语言。CPL 相比于 ALGOL 60 更接近硬件，但规模较大，难于实现。

1967 年英国剑桥大学的 Martin Richards 对 CPL 语言进行了简化，推出了 BCPL（Basic Combined Programming Language）语言。

1970 年美国贝尔实验室的 Ken Thompson 在 B 语言的基础上设计出 C 语言（取 BCPL 的第二个字母），既采用 B 语言的精炼和接近硬件的优点，又克服了过于简单、无数据类型等缺点。

1973 年，Ken Thompson 和 Dennis Ritchie 合作，用 C 语言改写了 UNIX 操作系统。到 1977 年，UNIX 操作系统得到日益广泛的使用，同时 C 语言也迅速得到推广。

1978 年 Brian W. Kernighan 和 Dennis M. Ritchie（合称 K&R）合著了影响深远的 *The C Programming Language* 一书。该书中介绍的 C 语言称为标准 C。

1983 年，美国国际标准化协会（ANSI）在参考 C 语言的各种版本的基础上，制定了新的标准，称为 ANSIC。

1987 年，ANSI 又公布了 C 语言新标准——87 ANSI C。1990 年，国际标准化组织接受了 87 ANSI C 为 C 语言的国际标准。目前流行的 C 编译系统都是以它为基础的。

高级语言发展至今，受到人们青睐的面向对象程序设计语言有 C++、Java、C#、Python 等。

1.2 第一个 C 语言程序——Hello，Word!

首先给出一个简单的实例，对 C 语言源程序有一个初步的认识。

【例 1.1】编写一个 C 程序，在屏幕上显示“Hello，Word!”。

```
#include <stdio.h>
main()                                  /*主函数*/
{
    printf("Hello,Word! \n");           /*输出信息*/
}
```

程序运行结果：

```
Hello,Word!
```

程序说明：

（1）该程序只由一个主函数构成，程序的第一行是文件包含命令行（文件包含内容将在后面章节介绍），第 2 行为主函数名，函数名后面的一对圆括号“（）”内用来添加函数的参数。参数可以有，也可以没有，但圆括号不能省略。

（2）程序中“{}”内的程序称为函数体，函数体通常由一系列语句组成，每一个语句用分号结束。

（3）函数中的 printf（）是系统提供的标准输出函数，可在程序中直接调用，其功能是把指定的内容显示到屏幕上。双引号内的“\n”表示换行，在信息输出后，光标将定位在屏幕下一行。

（4）“/*”和“*/”之间的文字是注释内容，不参与程序运行，目的是提高程序的可读性。

1.3 C 语言程序的基本组成

【例 1.2】编写一个 C 程序，计算并输出两个整数的和。

```
#include <stdio.h>
main()
{
    int a,b,sum;                        /*定义 3 个整型变量,分别存放两个整数和它们的和*/
    a=25;
    b=30;
    sum=a+b;
    printf("sum=%d\n",sum);
}
```

程序运行结果：

```
sum=55
```

程序说明：

（1）该程序的功能是求两个整数之和。

（2）函数体中首先定义了 3 个整型变量 a、b、sum，其中 int 表示整数类型；a、b、sum 为 3 个变量的名称，然后分别给变量 a、b 赋值，并将 a、b 之和赋给 sum。

（3）用 printf（）输出两个整数之和 sum。

【例 1.3】 从键盘输入两个整数，计算并输出它们的和。

```
#include <stdio.h>
/*add()函数用于求两个数之和*/
int   add(int x,int y)                    /*函数定义部分,add 为函数名,x,y 为形式参数*/
{
    int z;
    z=x+y;                                /*将两函数之和返回到主调函数中*/
    return(z);
}
/*main()函数完成两个整数的输入,并输出两数之和*/
main()
{
    int a,b,sum;
    printf("请输入两个数:");              /*输入两个整数,分别放入变量 a,b 中*/
    scanf("%d,%d",&a,&b);
    sum=add(a,b);                         /*调用 add()函数,将返回值赋给变量 sum*/
    printf("sum=%d\n",sum);
}
```

程序运行结果：

```
请输入两个数:5,7
sum=12
```

程序说明：

（1）该程序由主函数 main（）和被调用函数 add（）组成，它们各有一定的功能。main（）函数中的 scanf（）是系统的标准输入函数，其功能是输入 a 和 b 的值。

（2）本程序中提到函数调用、形式参数等概念，读者不必深究，在后续章节中会学习到相关知识。

根据前面列举的 3 个 C 程序，可以看出 C 语言源程序的基本组成形式。

1. C 语言程序是由函数组成的

函数是 C 语言程序的基本模块单元，每个函数完成相对独立的功能。每个程序必须有且只能有一个主函数 main（），除主函数外，可以没有其他函数（如例 1.1 和例 1.2），也可以有一个或多个其他函数（如例 1.3）。被调用的函数可以是系统提供的函数（如 ptintf（）和 scanf（）），也可以是用户根据需要自己编写的函数（如 add（））。

2. 函数的构成

一个函数由两部分组成：函数的首部和函数体。以例 1.3 中 add（）函数为例：

```
int   add(int x,int y)                    /*函数首部*/
{                                         /*函数体开始*/
    int z;                                /*函数体声明部分*/
```

```
    z=x+y;                                  /* 函数体执行部分 */
    return(z);
}                                           /* 函数体结束 */
```

函数首部是定义一个函数的开始，包括函数类型、函数名、函数形式参数表。函数名后必须跟上一对圆括号，函数可以没有参数。最简单的函数首部的形式如 main（）。

函数体是函数首部下面用一对大括号括起的部分，是函数功能的具体实现，包括声明部分和执行部分。声明部分中定义在本函数内部使用到的变量，此部分还可能对所调用到的函数进行声明；执行部分由 C 语句构成，用来完成函数所要实现的功能。

因此，一般来说，函数定义格式如下：

```
函数类型  函数名(形式参数表)
{
    声明部分
    执行部分
}
```

但根据实际情况，声明部分是可以没有的，甚至连执行部分也可以没有。例如：

```
Blankfun( )
{
}
```

这是一个空函数，什么也不做，没有意义，但是合法。

3. C 语言程序的执行

程序的执行总是从主函数开始，并在主函数结束的。主函数的位置是任意的，可以在程序的开头、两个函数之间或程序的结尾。

4. C 语言程序的书写格式

C 语言严格区分大小写。一般用小写字母书写，只有符号常量或其他特殊用途的符号才使用大写字母，所有关键字必须小写，如 if、else 、int 等必须小写。

注意：

If 不是关键字，Sum 和 sum 不是同一变量。

C 语言语句都必须以分号（;）结束。C 语言程序书写格式自由，允许一行内写多个语句，也可以一个语句写在多行。除非是程序流程控制语句（如 while 语句）等较复杂的语句，较简单的语句建议不要分行写。

可以用“/*…*/”对 C 语言程序的任何部分作注释，以增强程序的可读性。VC++ 中还可以用“//”给程序加注释，两种的区别在于“/*…*/”可以对多行进行注释，而“//”只能对单行进行注释。编译源程序时，不对注释作任何处理。注释通常放在一段的开始，用以说明该段程序的功能；或者放在某个语句的后面，对该语句进行说明。在使用“/*…*/”加注释时，需要注意“/*”和“*/”必须成对使用，且“/”和“*”以及“*”和“/”之间不能有空格，否则程序会出错。

1.4 VC++ 6.0 集成开发环境

集成开发环境是一个综合性的工具软件，它把程序设计过程中所需的各项功能有机地结合起来，统一在一个图形化操作界面下，为程序设计人员提供尽可能高效、便利的服务。

VC++ 6.0 就是一个功能齐全的集成开发环境，虽然它常常用来编写 C++源程序，但它同时兼容 C 语言程序的开发。

下面以例 1.1 为例，说明使用 VC++ 6.0 集成开发环境运行一个 C 语言程序的操作过程。

1. 启动 VC++ 6.0 环境

进入 VC++ 6.0 环境的方法有多种，最常用的方法：选择 Windows 操作系统的“开始 | 程序 | Microsoft Visual Studio 6.0 | Microsoft Visual C++6.0”命令，进入 VC++ 6.0。

VC++ 6.0 启动后，主窗口界面如图 1-4-1 所示。

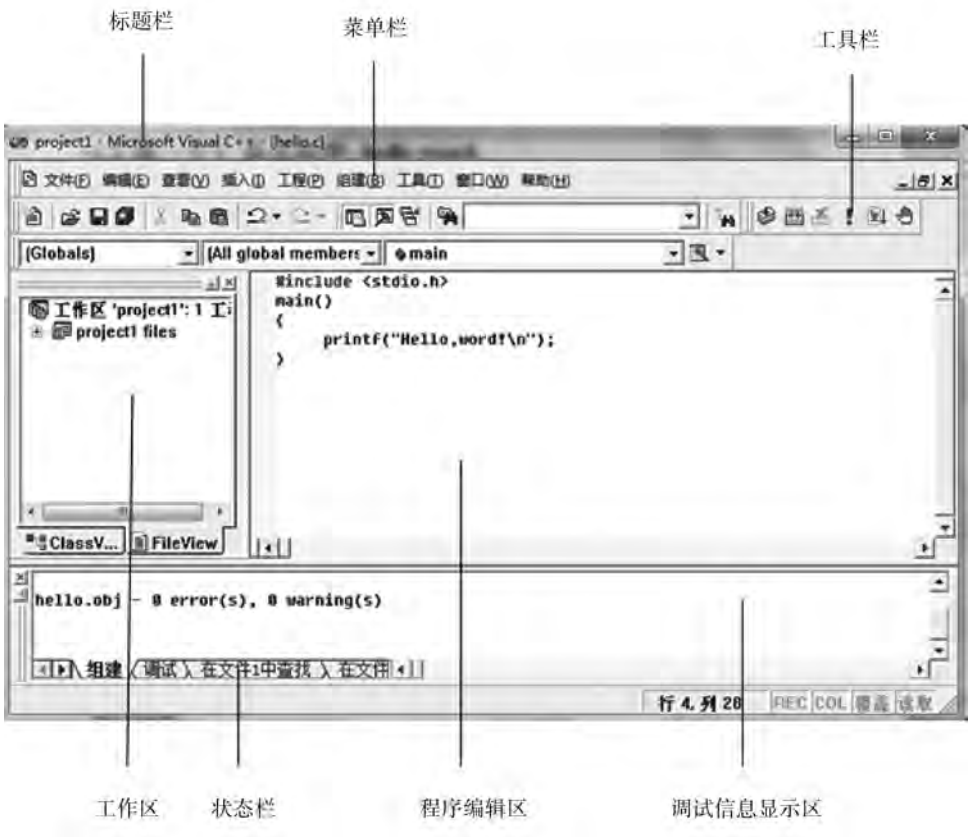


图 1-4-1 VC++6.0 主窗口界面

VC++ 6.0 主窗口和一般的 Windows 窗口并无太大的区别，由标题栏、菜单栏、工具栏、工作区、程序编辑区、调试信息显示区和状态栏组成。在没有编辑源程序的情况下，工作区无信息显示，程序编辑区为深灰色。

2. 编辑源程序文件

（1）建立新工程。

1）在图 1-4-1 所示的主窗口中，选择“文件 | 新建”命令，打开如图 1-4-2 所示的“新建”对话框。

图 1-4-2　“新建”对话框

2）在图 1-4-2 所示的“工程”选项卡左侧的工程类型中选择 Win32 Console Application 选项，在“工程名称”文本框中输入工程名称，如 project1；在“位置”文本框中输入或选择工程所存放的位置，单击“确定”按钮，弹出如图 1-4-3 所示对话框。

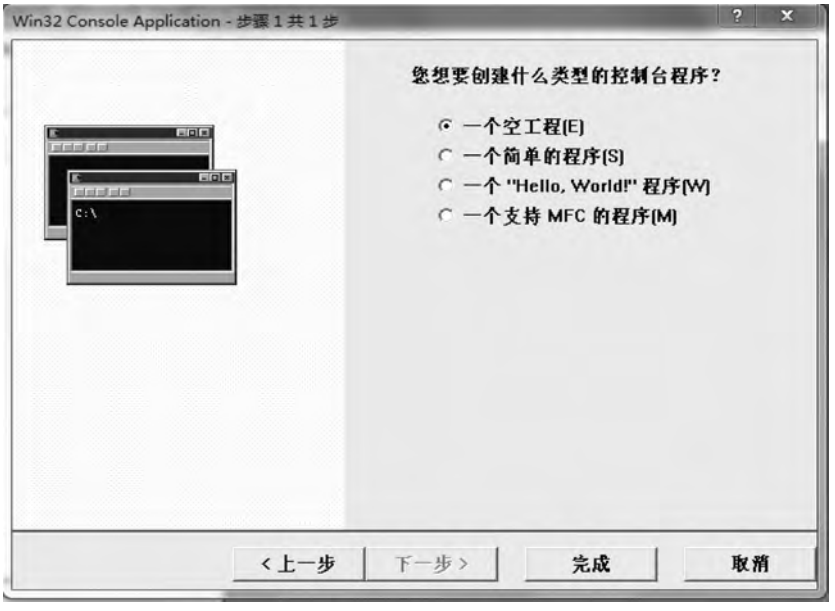


图 1-4-3　选择工程类型

3）在图 1-4-3 所示对话框中，选中“一个空工程”单选按钮，单击“完成”按钮。系统会弹出如图 1-4-4 所示的“新建工程信息”对话框，单击“确定”按钮，即完成了一个工程框架的创建。

图 1-4-4　“新建工程信息”对话框

（2）建立工程中的文件（也可以不建工程，直接用此步骤以单文件的方式建立源程序文件）。

1）在图 1-4-1 所示的主窗口中，选择“文件 | 新建”命令，弹出如图 1-4-2 所示的“新建”对话框。

2）在“文件”选项卡左侧的文件类型中选择 C++ Source File 选项，在“文件名”文本框中输入文件名，如 hello. c（注意：由于编写的是标准 C 语言程序，应加上文件的扩展名 . c，否则系统会自动取默认的扩展名 . cpp），如图 1-4-5 所示。单击“确定”按钮，则创建了一个源程序文件，并返回到图 1-4-1 所示的 VC++ 6. 0 主窗口。

3）在主窗口程序编辑区输入例 1. 1 中的源程序。

3. 编译

方法一：选择主窗口菜单栏中的“组建 | 编译［hello. c］”命令，进行编译。

方法二：单击主窗口编译工具栏上的按钮 进行编译。

在编译过程中，系统如发现程序有语法错误，则在调试信息显示区显示错误信息，并给出错误性质、出错位置和错误原因等。用户可通过双击某条错误来确定该错误在源程序中的具体位置，并根据出错性质和原因对错误进行修改。修改后再重新进行编辑，直到没有错误信息为止。

图 1-4-5　建立工程中的文件

编译出错信息有两类：一类是 error，说明程序肯定有错，必须修改；二是 warning，表明程序可能存在潜在的错误，只是编译系统无法确定，希望用户检查。对于第二类出错信息，如果用户置之不理，也可生成目标文件，但存在运行风险，因此，建议把 warning 当成 error 来严格处理。

4. 链接

编译无错误后，可进行链接，生成可执行文件。

方法一：选择主窗口菜单栏中的“组建 | 组建［hello. exe］”命令，进行链接。

方法二：单击主窗口编译工具栏上的按钮 进行链接。

编译、链接成功后，即在当前工程文件夹下生成可执行文件（hello. exe）。

5. 运行

方法一：选择主窗口菜单栏中的“组建 | 执行［hello. exe］”命令，执行编译、链接后的程序。

方法二：单击主窗口编译工具栏上的按钮 ，执行编译、链接后的程序。

若程序运行成功，屏幕上将输出运行结果，并给出提示信息 Press any key to continue，表示程序运行后，可按任意键返回 VC++6. 0 主窗口。运行结果如图 1-4-6 所示。

图 1-4-6　运行结果窗口

1.5 C 语言程序的编写和运行步骤

开发 C 语言程序是指在一个集成开发环境中对程序进行编辑、编译、链接和运行的过程。

（1）编辑：程序员使用编辑软件，如写字板、记事本或集成化的程序设计软件等编写的 C 语言程序称为 C 源程序（文件扩展名为 .c，但在 VC++ 6.0 中，扩展名为 .cpp）。

（2）编译：C 源程序经由编译器转换成机器代码，生成扩展名为 .obj 的目标文件。在编译过程中，如果程序存在错误，则返回编辑状态进行修改。

（3）链接：C 语言是模块化的程序设计语言，一个 C 语言应用程序可能由多个程序设计者分工合作完成，需要将所有用到的库函数及其他目标程序链接为一个整体，生成扩展名为 .exe 的可执行文件。

（4）运行：运行可执行文件后，可获得程序运行结果。如果运行后没有达到预期目的，则需进一步修改源程序，重复上述过程，直到达到设计要求。

1.6 VC++ 6.0 环境下的程序调试

在一个 C 源程序编辑完成之后，可以用上一节讲的步骤进行编译、链接、运行。如果程序编译错误，或运行结果有误时，就需要调用 VC 的编译调试工具来调试程序，以便找到出错的原因，从而得到正确的运行结果。

1. 修改语法错误

查看运行结果时要注意错误的类别，是语法错误还是算法逻辑错误。对于语法错误，可按照系统所提示的错误类型对错误进行修改。

操作步骤：双击显示错误或警告的第一行，则光标自动跳到错误代码处。切记，一定要先修改出现的第一个语法错误。修改了错误后，重新进行编译。一般在第一个错误修改后，后面与其有关的错误自然就解决了。若还有错误，仍然是先修改第一个错误，然后再次进行编译，直到没有错误和警告信息为止。

2. 编辑、运行与调试程序中常用到的快捷键

Ctrl+O：打开文件；

Ctrl+S：保存文件；

Ctrl+F7：编译文件；

F7：链接文件；

Ctrl+F5：运行文件；

F9：将光标所在行位置设置为断点；

F5：调试运行到断点；

F10：不进入函数内部调试；

F11：进入函数内部调试；

Shift+F5：中断调试。

3. 修改算法错误

程序编译语法通过，但运行结果不正确，就要查找、分析算法，重新进行程序设计。

在程序修改后，还要重新进行编译、链接、运行，生成 .exe 文件后再次运行，直到结果正确。C 语言编程不仅要严格掌握语法规则，还要认真进行算法分析和设计。

任务二　实用计算器中的数据类型与运算

学习目标

- 了解 C 语言的基本字符、标识符和关键字；
- 掌握 C 语言的编程规范；
- 掌握 C 语言的基本数据类型、常量和变量；
- 掌握 C 语言的运算符、表达式及其使用方法。

一、任务描述

在任务一的基础上，实现实用计算器中数据的运算功能。从键盘输入两个运算数，按加、减、乘、除顺序依次进行运算，并输出对应的运算结果。

二、知识要点

本任务中涉及数据类型、常量和变量、运算符和表达式等知识。

三、任务分析

根据系统的功能设计出系统操作界面。

（1）显示系统操作菜单。

（2）从键盘输入运算数。选用 C 语言提供的标准输入函数 scanf（）。

（3）加、减、乘、除四种运算的实现。四种运算分别用算术运算符“+”“-”“*”“/”实现。在书写程序时，需要注意 C 语言中的乘、除表示方法和数学公式中的不同，最后的运算结果可用 printf（）函数输出。

四、源代码参考

```
#include <stdio.h>
void main()
{
    float data1,data2;                    //存放两个运算数
    printf("\n\n");
    printf("\t\t|--------------------------------------|\n");
    printf("\t\t|              实用计算器              |\n");
    printf("\t\t|--------------------------------------|\n");
    printf("\t\t|              1—加法                  |\n");
    printf("\t\t|              2—减法                  |\n");
```

```
    printf("\t\t|                3—乘法                                |\n");
    printf("\t\t|                4—除法                                |\n");
    printf("\t\t|                0—退出                                |\n");
    printf("\t\t|--------------------------------------------------|\n");
    printf("\t\t  请输入第一个运算数:");
    scanf("%f",&data1);
    printf("\t\t  请输入第二个运算数:");
    scanf("%f",&data2);
    printf("\t\t  加法运算结果为:\n");
    printf("\t\t  %f + %f = %f \n",data1,data2,data1+data2);
    printf("\t\t  减法运算结果为:\n");
    printf("\t\t  %f - %f = %f \n",data1,data2,data1-data2);
    printf("\t\t  乘法运算结果为:\n");
    printf("\t\t  %f * %f = %f \n",data1,data2,data1*data2);
    printf("\t\t  除法运算结果为:\n");
    printf("\t\t  %f / %f = %f \n",data1,data2,data1/data2);
}
```

2.1　数据类型的分类

C 语言中，经常用到的各种类型数据如图 2-1-1 所示。

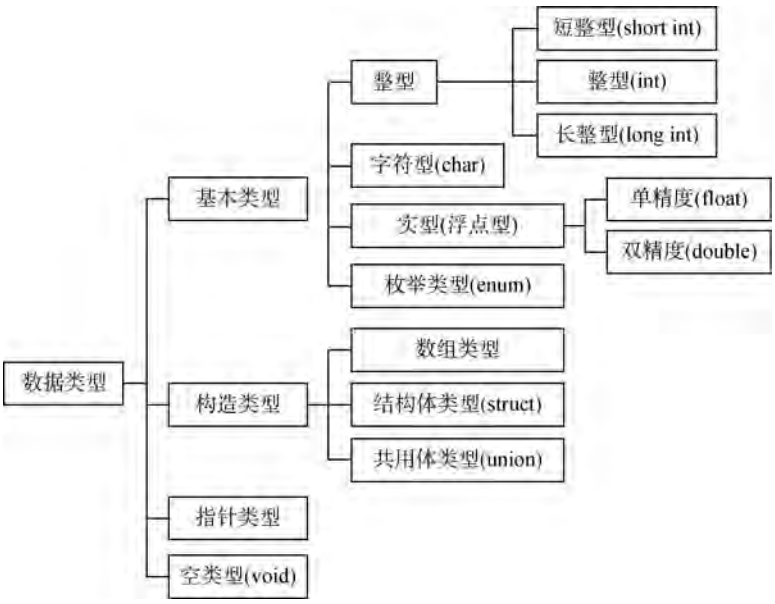


图 2-1-1　数据类型

C 语言基本数据类型见表 2-1-1。

表 2-1-1　C 语言基本数据类型

数据类型	所占空间/B	取值范围
布尔型（bool）	1	false 或 true
有符号字符型（char 或 signed char）	1	-128~127
无符号字符型（unsigned char）	1	0~255
有符号短整型（short 或 short int）	2	-32 768~32 767
无符号短整型（unsigned short 或 unsigned short int）	2	0~65 535
有符号整型（int 或 signed int）	4	-2^{31} ~ （$2^{31}-1$）
无符号整型（unsigned 或 unsigned int）	4	0 ~ （$2^{32}-1$）
有符号长整型（long 或 long int 或 signed long int）	4	-2^{31} ~ （$2^{31}-1$）
无符号长整型（unsigned long 或 unsigned long int）	4	0 ~ （$2^{32}-1$）
浮点型（float）	4	-3.4×10^{38}~3.4×10^{38}
双精度型（double）	8	-1.7×10^{308}~1.7×10^{308}
长双精度型（long double）	12	$-1.2\times10^{4\,932}$~$1.2\times10^{4\,932}$

说明：

ISO C/C++语言标准并没有明确规定每种数据类型的字节数和取值范围，只是规定它们之间的字节数大小顺序，因此不同的编译器对此会有不同的实现。现在国际上常用的 ISO C/C++语言标准有 ISO IEC C99、ISO IEC C11、ISO IEC C++11、ISO IEC C++14、ISO IEC C++17。

2.2　常量

常量是在程序执行过程中其值不能被修改的量，分为字面常量和符号常量。字面常量的类型是根据书写形式来区分的，例如 1、2.3、-4.5、'h'，"C++" 等等都是字面常量。符号常量是一个标识符，指代某一个常量。

2.2.1　整型常量

整型常量按进制分为三种形式：

（1）十进制数：以正号（+）或符号（-）开头，以首位非 0 的一串数字组成。如 1、-2 等。

（2）八进制数：以数字 0 开头，后面跟一串数字（数字只能取 0~7），如 0123 表示十进制数 83。

（3）十六进制数：以数字 0 和字母 x（或 X）开头，后面接一串数字（数字只能取 0~9，字母取 A~F 或 a~f，代表 10~15），如 0X23、0X2A4，对应十进制整数依次为 35、676。

下面的形式是错误的：

079

因为 9 不在八进制的基数范围内。

2.2.2 实型（浮点型）常量

C 语言的实型常量表示形式有 float 和 double 两种，默认实型常量为 double 型，如果要表示 float 型，要在数字后加 f 或 F，如 1.23F。

（1）十进制数表示形式：1.23，1.23F 等。

（2）科学计数法表示形式：1.2e2 或 1.2e+2，相当于 1.2×10^2，其中 1.2 为数字部分，2 是指数部分。C 语言用字母 e（或 E）表示其后的数是以 10 为底的幂，如 e2 相当于 10^2，同时 C 语言规定 e 或 E 之前必须有数字，且后面的指数必须为整数。

2.2.3 字符型常量

字符型常量都是用单引号括起来的单个字符，某些不能用引号括起来直接表示的字符，可以使用转义字符' \ ' 来实现，如用' \ ' 来代表单引号本身。此外，还可以后跟 1~3 个八进制数，如' \ 141' 代表字符' a' 。C 语言预定义的转义字符见表 2-2-1。

表 2-2-1 C 语言预定义的转义字符

转义字符	描述	转义字符	描述
\ ddd	1~3 位八进制数表示的字符	\ xhh	1~2 位十六进制数表示的字符
\ '	单引号	\ "	双引号
\ r	回车，将当前位置移到本行开头	\ \	反斜线
\ n	换行，将当前位置移到下一行开头	\ b	退格，将当前位置移到前一列
\ t	水平制表位（跳到下一个 Tab 位置）	\ v	垂直制表符（竖向跳格）
\ f	换页，将当前位置移到下页开头	\ a	响铃

说明：

转义字符在内存中以 ASCII 码的形式存储，每个转义字符只代表一个字符，在内存中只占 1 个字节。

2.2.4 字符串常量

字符串常量是用一对双引号括起来的字符序列。例如：""guangjiu""he yin chuan""等都是字符串常量。字符串在内存中是按串中字符的排列次序顺序存放，每个字符占一个字节，并在末尾添加' \ 0' 作为结尾标记。

例如：字符串""hello""在内存中实际存放形式如图 2-2-1 所示。其长度为 6 个字节，而不是 5 个字节。字符' \ 0' 对应的 ASCII 码值为 0，即“空”字符。

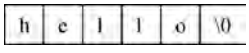


图 2-2-1 字符串的存储形式

' a' 与" a" 是不同的，' a' 是字符常量，内存中只存储字符 a 的 ASCII 码值，所以只占 1 个字节。""a""是字符串常量，内存中存储字符 a 和' \ 0' ，一共占 2 个字节。

注意：

C 语言中，字符串常量没有对应的字符串变量。字符串如果在变量中保存，需要用字符型数组来保存。关于字符型数组，将在项目二介绍。

2.2.5 布尔常量

布尔常量只有两个：true（真）和 false（假）。

2.2.6 符号常量

在 C 语言中通常用标识符代表常量，代表常量的标识符称为符号常量。符号常量一般用大写字母表示，如 PRICE、PI。

语法格式如下：

```
#define  符号常量  常量
```

其值在作用域内不能改变和再赋值。符号常量的优点是见名知意，一改全改。

【例 2.1】符号常量举例。

```
#define  PRICE  30
#include <stdio.h>
void main()
{
    int num,total;
    num=10;
    total=num * PRICE;
    printf("total=%d\n",total);
}
```

程序运行结果为：

```
total=300
```

2.3 变量

变量即在程序运行过程中其值是允许改变的量。变量必须说明，说明的目的是让程序知道变量类型并给变量分配相应数量的存储单元。变量必须先说明，后使用，并且变量名不可为系统保留字。

2.3.1 标识符

标识符是对变量、数组名、函数名、类和对象等的命名标志。在 C 语言中，标识符的命名规则如下：

（1）只能由字母、数字或下画线“_”组成，且不能是关键字；

（2）首字符不能是数字，只能是字母或下画线；

（3）大小写敏感，大写和小写代表不同的标识符，同时一般用小写字母表示变量名；

（4）没有长度限制，但有的系统只取前 32 个字符，建议命名时不超过 32 个字符。

下面举例说明 C 语言中标识符的使用规则。

（1）合法标识符：

char a_3；float_var3；

double money。

（2）不合法的标识符：

int 3a；（不能以数字开头）

char abc-8；（下画线“_”可以，但横线“-”不可以作为标识符字符）

char p3@；（有非法字符@）

int for。（不能是关键字）

注意：

标识符应该“见名知意”，如：total，max。

标识符应该“不宜混淆”，如：l 与 1，O 与 0。

2.3.2 关键字

C 语言本身保留了一些特殊的标识符，称为关键字或保留字。关键字有着特定的语法含义，不同的使用目的，它们不允许被定义为普通的标识符。C 语言中的关键字都用小写字母表示。表 2-3-1 列出了 C 语言中使用的关键字。

表 2-3-1 C 语言的关键字

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

1999 年 12 月 16 日，ISO 推出了 C99 标准，该标准新增了 5 个 C 语言关键字，见表 2-3-2。

表 2-3-2 新增关键字 1

inline	restrict	_Bool	_Complex	_Imaginary

2011 年 12 月 8 日，ISO 发布 C 语言的新标准 C11，该标准新增了 7 个 C 语言关键字，见表 2-3-3。

表 2-3-3 新增关键字 2

_Alignas	_Alignof	_Atomic	_Static_assert	_Noreturn	_Thread_local	_Generic

2.3.3 变量的定义

变量是在程序执行过程中可根据需要经常变化的值，可以为每个变量指定名称以便编译器可唯一标识。

变量由以下三部分组成：

（1）名称：标识符。

（2）初始值：为其赋值或者是保留缺省值。

（3）作用域：在不同程序块中的可用性及生命周期。

C 语言规定变量要先定义后使用，目的是为变量分配空间，同时为了编译时进行与操作相关的语法检查。变量定义的语法格式如下：

<数据类型><变量名 1>[=初始值 1],……,<变量名 n>[=初始值 n];

其中，方括号表示可选项，尖括号表示必选项，变量名要遵守标识符命名规则。

例如：

```
int i;
i=8;//先定义整型变量 i,再给 i 赋初值 8,其在内存中的存储形式如图 2-3-1 所示
```

图 2-3-1　变量在内存中的存储形式

上面 2 条语句也可以写作 int i=8；这种在定义变量的同时进行赋值，称为变量的初始化。

变量可分为全局变量和局部变量。全局变量指具有类块作用域的类成员变量，局部变量指具有方法块作用域的变量。局部变量必须初始化或赋值，否则不能使用。全局变量的默认初始值为该变量数据类型的默认值，见表 2-3-4。

表 2-3-4　类成员变量的默认值

类成员变量的数据类型	默认值
布尔类型（bool）	false
整型（int）	0
实型（float）	0.0
字符型（char）	'\0'

1. 整型变量

整型变量数据在内存中以二进制补码形式存放，如图 2-3-2 所示。其定义方式为：

```
int a,b;                          //指定变量 a、b 为整型
unsigned short c,d;               //指定变量 c、d 为无符号短整型
long e,f;                         //指定变量 e、f 为长整型
```

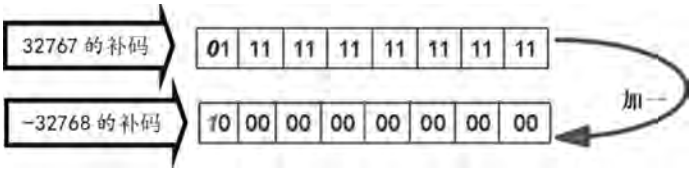


图 2-3-2　整型数据在内存中的存放形式

注意：

当整型变量为最大值32767时，再加1后是以-32768的补码形式存放在内存中，此情况称为“溢出”，运行时不报错，编程时要注意。

2. 实型变量

float型数据在内存中占4个字节（32位），double型数据在内存中占8个字节（64位）。实型数据在内存中分成3部分，如图2-3-3所示。其定义方式为：

```
float x,y;                    //指定x、y为单精度浮点型变量
double z;                     //指定z为双精度浮点型变量
long double t;                //指定t为长双精度浮点型变量
```

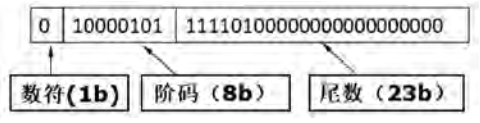


图2-3-3 实型数据在内存中的存放形式

3. 字符型变量

每个字符型变量只能存放一个字符。其定义方式为：

```
char cr1,cr2;                 //指定cr1、cr2为字符型变量
char x='a';                   //指定x为字符型变量,存储了字符'a'
```

字符型数据与整型数据之间可以通用，一个字符能用字符的形式输出，也能用整数的形式输出。字符数据进行算术运算时，相当于对它们的ASCII码进行运算。

2.4 运算符和表达式

表达式是C语言的主体，在C语言中，表达式由操作符（即运算符）和操作数组成。简单的表达式可以只有一个操作数。根据操作符的个数，可以将表达式分为简单表达式和复杂表达式。简单的表达式只含有一个操作符（如5+5），而复杂的表达式则含有两个或者两个以上的操作符（如5+5+6*4）。表达式本身什么也不做，只返回结果的值，在程序不对结果值做任何处理的情况下，返回的结果值不起任何作用。表达式的用法有两种：①放在赋值语句的右侧；②放在函数的参数中。

2.4.1 运算符和表达式概述

运算符是对操作数进行运算的符号。按操作符所操作的数目来分，运算符可以分为一元（单目）运算符、二元（双目）运算符和三元运算符。其中一元运算符分为前置和后置两种。如果按照运算功能来分，运算符可以分为下面几类：

（1）算术运算符（+、-、*、/、%、++、--）；

（2）关系运算符（>、<、>=、<=、==、!=）；

（3）位运算符（〉〉、〈〈、&、|、^、~）；

（4）逻辑运算符（!、&&、||）；

（5）赋值运算符（=及其扩展赋值运算符）；

（6）条件运算符（?:）；

（7）其他运算符（逗号运算符、指针运算符、引用运算符、地址运算符、求字节运算符、强制类型转换运算符、类或实例成员操作运算符、指向成员的运算符和下标运算符等）。

本节主要讲解前6种运算符，其他运算符将在后续章节中逐个讲解。

表达式就是由运算符将运算对象按照一定的语法规则连接起来的式子。表达式由常量、变量、函数、运算符、括号等内容组成。表达式的求值顺序为：若表达式有圆括号运算符，先计算括号内的值，再计算括号外的值；若表达式有多个运算符，则按照运算符优先级顺序计算，如果优先级也相同，则按运算符的结合性进行计算。运算符的优先级与结合性见表2-4-1。

表2-4-1　运算符的优先级与结合性

优先级	运算符	名称或含义	使用形式	结合方向	说明
1	[ ]	数组下标	数组名［整型表达式］	从左到右	
	( )	圆括号	（表达式）/函数名（形参表）		
	.	成员选择（对象）	对象．成员名		
	->	成员选择（指针）	对象指针->成员名		
2	-	负号运算符	-算术类型表达式	从右到左	一元运算符
	(type)	强制类型转换	（纯量数据类型）纯量表达式		
	++	自增运算符	++纯量类型可修改左值表达式		一元运算符
	--	自减运算符	--纯量类型可修改左值表达式		一元运算符
	*	取值运算符	* 指针类型表达式		一元运算符
	&	取地址运算符	& 表达式		一元运算符
	!	逻辑非运算符	! 纯量类型表达式		一元运算符
	~	按位取反运算符	~整型表达式		一元运算符
	sizeof	长度运算符	sizeof 表达式 sizeof（类型）		
3	/	除	表达式/表达式	从左到右	二元运算符
	*	乘	表达式 * 表达式		二元运算符
	%	余数（取模）	整型表达式%整型表达式		二元运算符
4	+	加	表达式+表达式	从左到右	二元运算符
	-	减	表达式-表达式		二元运算符

续表

优先级	运算符	名称或含义	使用形式	结合方向	说明
5	<<	左移	整型表达式<<整型表达式	从左到右	二元运算符
	>>	右移	整型表达式>>整型表达式		二元运算符
6	>	大于	表达式>表达式	从左到右	二元运算符
	>=	大于等于	表达式>=表达式		二元运算符
	<	小于	表达式<表达式		二元运算符
	<=	小于等于	表达式<=表达式		二元运算符
7	==	等于	表达式==表达式	从左到右	二元运算符
	!=	不等于	表达式!= 表达式		二元运算符
8	&	按位与	整型表达式 & 整型表达式	从左到右	二元运算符
9	^	按位异或	整型表达式^整型表达式	从左到右	二元运算符
10	|	按位或	整型表达式| 整型表达式	从左到右	二元运算符
11	&&	逻辑与	表达式 && 表达式	从左到右	二元运算符
12	||	逻辑或	表达式|| 表达式	从左到右	二元运算符
13	?:	条件运算符	表达式 1? 表达式 2：表达式 3	从右到左	三元运算符
14	=	赋值运算符	可修改左值表达式=表达式	从右到左	
	/=	除后赋值	可修改左值表达式/=表达式		
	*=	乘后赋值	可修改左值表达式 *=表达式		
	%=	取模后赋值	可修改左值表达式%=表达式		
	+=	加后赋值	可修改左值表达式+=表达式		
	-=	减后赋值	可修改左值表达式-=表达式		
	<<=	左移后赋值	可修改左值表达式<<=表达式		
	>>=	右移后赋值	可修改左值表达式>>=表达式		
	&=	按位与后赋值	可修改左值表达式 &=表达式		
	^=	按位异或后赋值	可修改左值表达式^=表达式		
	|=	按位或后赋值	可修改左值表达式| =表达式		
15	,	逗号运算符	表达式，表达式，…	从左到右	从左向右顺序结合

2.4.2 算术运算符和算术表达式

1. 算术运算符

算术运算符分一元运算符、二元运算符和三元运算符。一元运算符只有一个参与运算的操作数，二元运算符有两个参与运算的操作数。表 2-4-2 列出了算术运算符及其用途和相关说明。

表 2-4-2　算术运算符及其用途

运算符	用途	例子	说明
+，-	加、减	b=a-5	把变量 a 减去 5，结果赋值给 b
+，-	取正、取负	b=-a;	把 a 取负后赋值给 b
*，/	乘、除	a=3*4/5.0	3*4 为 12，12/5.0 为 2.4，赋值给 a
%	求余	a=6%4	C 语言中% 要求左右两侧的操作数都是整数，6%4 结果为 2，赋值给 a
++，--	自增 1、自减 1	i++，--j	i 先参与运算，再自动加 1 赋值给自己；j 先自动减 1 赋值给自己，然后参与运算

注意：

自增、自减运算符为++ 、--

作用：使变量值加 1 或减 1

种类：

前置　++i，--i　（先执行 i+1 或 i-1，再使用 i 值）

后置　i++，i--　（先使用 i 值，再执行 i+1 或 i-1）

```
例如：j=3;    k=++j;                  //k=4，j=4
      j=3;    k=j++;                  //k=3，j=4
      j=3;    printf ("%d", ++j);     //4，j=4
      j=3;    printf ("%d", j++);     //3，j=4
      a=3; b=5; c= (++a) *b;          //c=20，a=4
      a=3; b=5; c= (a++) *b;          //c=15，a=4
```

算术运算符在使用过程中易出现一些常见的错误，见表 2-4-3。

表 2-4-3　算术运算符中常见的错误

常见的错误	说　明
int x=7%3.0;	%要求左右两侧的操作数都是整数
int x=8/3;	两个整数相除，结果取整数并且不进行四舍五入运算，x 为 2 而不是 2.7
float f=2f+3.0;	3.0 为 double 型，2f+3.0 返回值为 double 型

2. 算术表达式

用算术运算符和括号将运算对象连接起来，且符合 C 语言语法规则的式子称为算术表达式。在进行算术表达式运算时，要注意算术运算符的优先级和结合性。一般情况下，算术运算符的优先级是先乘除后加减；当运算符优先级相同时，按照算术运算符“自左向右”的结合方向进行运算。

【例 2.2】自增自减一元算术运算符举例。

```
#include <stdio.h>
main()
{
    int i=4,j=4,k,m;
    k=i++;  //i 的值 4 赋值给 k,然后 i 自动加 1
    m=--j;  //j 先自动减 1,然后把变化后的值赋值给 m
    printf("i=%d,j=%d,k=%d,m=%d\n",i,j,k,m);
    return 0;
}
```

程序运行结果：

```
i=5,j=3,k=4,m=3
```

2.4.3 关系运算符和关系表达式

1. 关系运算符

关系运算符主要用来比较两个运算对象的大小。C 语言中关系运算符有 6 种：>、<、>=、<=、==、!=，关系运算符用来把两个或者更多的运算对象连接起来，其中运算对象包含变量、常量或表达式。使用关系运算符时要注意以下两点：

（1）运算符>、<、>=、<=的优先级相同，但优先级大于==、!=。

（2）关系运算符的结合性是自左至右。关系运算符的优先级比赋值运算符高，但是比算术运算符低。

2. 关系表达式

关系表达式是用关系运算符把两个或多个运算对象连接起来的表达式。运算对象包含常量、变量或者表达式。关系表达式的运行结果为逻辑值，结果为“真”或“假”两种。其中用“0”表示假，用“1”表示真。

例如，a=1，b=2，c=3，有下列关系表达式：

```
a+b>2*c                    //结果为“0”
a+b==c                     //结果“1”
++a==b++                   //结果为“1”
```

2.4.4 位运算符

C 语言的位运算符有左移位运算符“<<”和右移位运算符“>>”。位运算符的操作数为整型数据。表 2-4-4 列出了位运算符的用途和相关说明。

表 2-4-4 位运算符的用途和相关说明

运算符	用途	举例	说 明
<<	按位左移	65<<2	01000001 按位左移 2 位，得 00000100
>>	按位右移	65>>1	01000001 按位右移 1 位，得 00100000
~	按位取反	~65	01000001 按位取反，得 10111110
&	按位与	65&66	01000001 和 01000010 按位与，得 01000000
|	按位或	65|66	01000001 和 01000010 按位或，得 01000011
^	按位异或	65^66	01000001 和 01000010 按位异或，得 00000011

2.4.5 逻辑运算符和逻辑表达式

逻辑运算符是用来进行逻辑运算的运算符。C 语言中逻辑运算符包括!、&& 和 | | 。运算符! 对应 NOT 运算，运算符 && 对应 AND 运算，运算符 | | 对应 OR 运算。表 2-4-5 列出了逻辑运算符及其用途。

表 2-4-5 逻辑运算符及其用途

运算符	用途	举例	说 明
!	取反运算	!（100>99）	100>99 为 true，取反后结果返回 false
&&	逻辑与运算	（8>7）&&（10>36）	&& 的左侧（8>7）为真，右侧（10>36）为假，相与的结果为假
| |	逻辑或运算	（3>4）| |（60>50）	| | 的左侧（3>4）为假，右侧（60>50）为真，相或的结果为真

说明：

（1）&& 的左侧为假，右侧则不再进行计算，结果为假。

（2）| | 的左侧为真，右侧则不再进行计算，结果为真。

2.4.6 赋值运算符和赋值表达式

1. 赋值运算符

赋值运算符“=”的左边是变量，右边是表达式，表达式的运算结果应和左边的变量类型一致，或能转换为左边变量的类型。

```
int   k=1+2;                    //正确
int   b=3.9+5;                  //右值为 8.9,是 double 型,但 b 是整型,取 8 赋值给 b
int   c=(int)(2.9+4);           //右值为 6.9,是 double 型,(int)(6.9)强制转换为整型,所以 c 为 6
double   d=k+6;                 //正确
```

可以看出，赋值时，应遵循数据转换规则。

赋值运算符“=”还可以同其他运算符相结合，实现运算和赋值双重功能，简称复合赋值运算符。C 语言一共提供了 10 种复合赋值运算符。分别是+=、-=、*=、/=、%=、&=、|=、^=、<<=和>>=。

```
a+=3                            //等价:a=a+3
x*=y+8                          //等价:x=x*(y+8)
x%=3                            //等价:x=x%3
```

2. 赋值表达式

将一个变量和表达式用赋值运算符连接起来的式子就是赋值表达式。赋值表达式的一般格式为：

```
<变量> <赋值运算符> <表达式>
```

例如：

```
int x=8,y;                      //这个赋值表达式就是将赋值号右边的 8 赋值给左边的变量 x。
y=(x=6);                        //x 的值为 6,y 的值也为 6,整个表达式的值也为 6。
a=(b=4)+(c=6)                   //表达式的值为 10,a=10,b=4,c=6
a=(b=10)/(c=2)                  //表达式的值为 5,a=5,b=10,c=2
```

2.4.7 条件运算符和条件表达式

1. 条件运算符

条件运算符（?:）是C语言中唯一的一个三元运算符，其是由问号“?”和冒号“:”组成的，连接三个运算对象，用来在两个表达式中选择一个表达式。

条件表达式的优先级：

（1）条件表达式的优先级高于赋值运算符、逗号运算符，低于其他运算符；

（2）条件运算符的结合顺序为“自右至左”。

2. 条件表达式

使用条件运算符把表达式连接起来的表达式叫条件表达式，一般表达形式为：

```
表达式1? 表达式2:表达式3;
```

表达式1是关系或布尔型，返回值为布尔型。如果表达式1的值为true，则整个表达式的值为表达式2的值。如果表达式1的值为false，则整个表达式的值为表达式3的值。

【例2.3】利用条件表达式求两个数的最小值。

```
#include <stdio.h>
void main()
{
    int x,y,min;
    printf("请输入x和y的值:");
    scanf("%d%d",&x,&y);
    min=x<y? x:y;
    printf("最小值是:%d\n",min);
}
```

程序运行结果为：

```
请输入x和y的值:4 3
最小值是:3
```

2.4.8 逗号运算符和逗号表达式

一般表达形式：

表达式1，表达式2，……，表达式n；

逗号运算符的运算顺序是从左到右，优先级为15，级别最低，常用于循环for语句中。整个逗号表达式的值是最后一个表达式的值。

例如：

```
a=3*5,a*4                    //a=15,表达式的值为60
a=3*5,a*4,a+5                //a=15,表达式的值为20
```

例如：

```
x=(a=3,6*3)                  //赋值表达式,表达式的值为18,x=18
x=a=3,6*a                    //逗号表达式,表达式的值为18,x=3
```

例如：

```
a=1;b=2;c=3;
printf("%d,%d,%d",a,b,c);          //输出结果为1,2,3
printf("%d,%d,%d",(a,b,c),b,c);    //输出结果为3,2,3
```

【例 2.4】逗号表达式的使用。

```
#include <stdio.h>
main()
{
    int x,y=7;
    float z=4;
    x=(y=y+6,y/z);
    printf("x=%d\n",x);
}
```

程序运行结果为：

```
x=3
```

2.4.9 各类数值型数据间的混合运算

在 C 语言中，整型、实型、字符型数据间可以进行混合运算，实际运算时先将不同类型的数据转换成同一类型后再进行运算。数据转换有两种形式，即自动转换和强制转换。

1. 自动转换

（1）运算转换——不同类型数据混合运算。

（2）赋值转换——把一个值赋给与其类型不同的变量。

（3）输出转换——输出时转换成指定的输出格式。

（4）函数调用转换——实参与形参类型不一致时转换。

运算转换规则如图 2-4-1 所示。

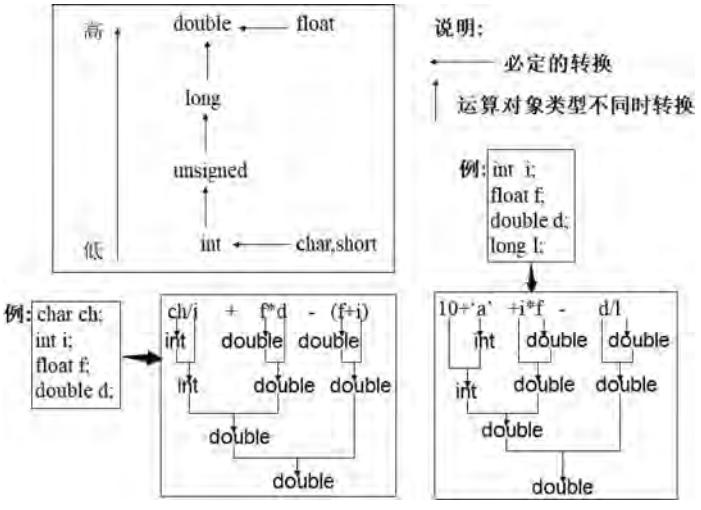


图 2-4-1 运算转换规则

2. 强制转换

一般形式：（类型名）（表达式）

例：

```
(int)(x+y)
```

```
(int)x+y
(double)(3/2)
(int)3.6
```

说明：强制转换得到所需类型的中间变量，原变量类型不变。

【例 2.5】强制转换的运用。

```
#include <stdio.h>
void main()
{
    float  x;
    int  i;
    x=3.6;
    i=(int)x;
    printf("x=%f,i=%d",x,i);
}
```

程序运行结果为：

```
x=3.600000,i=3
```

注意：

较高类型向较低类型转换时可能发生精度损失问题。

任务三　实用计算器中菜单的设计

学习目标

- 掌握顺序结构程序的编写方法；
- 熟练掌握选择结构、循环结构的设计方法；
- 掌握控制转移语句的使用方法。

一、任务描述

用选择语句实现实用计算器中选择执行运算功能。从键盘输入两个运算数，用户在系统操作菜单中选择运算类型对应的数字，按用户的选择进行运算，并输出对应的运算结果。

二、知识要点

本任务主要涉及 if…else 语句、switch 语句、for 语句、while 语句、do…while 语句等的使用。

三、任务分析

根据系统的功能设计出系统操作界面。

（1）显示系统操作菜单。

（2）加、减、乘、除和退出的选择，属于多分支选择结构。本任务使用了 switch 语句实现主菜单的选择，请试使用 if…else 语句实现。

（3）若要实现菜单的重复显示，需要用到循环结构。本任务只给出了 while 语句实现主菜单的重复显示，请试使用 for 语句、do…while 语句实现。

四、源代码参考

```
#include <stdio.h>
#include <stdlib.h>
void main()
{
    float data1,data2;              //存放两个运算数
    int choice;                     //存放用户选择的代表运算类型的数字
    char judge;                     //存放是否继续
    judge='y';
    while(judge=='y'||judge=='Y')   //也可用 toupper(judge)=='Y' 统一转换为大写字母,须加
                                      头文件 string.h
    {
        printf("\n\n");
        printf("\t\t|*************************************************************|\n");
        printf("\t\t|                      实用计算器                             |\n");
        printf("\t\t  |-----------------------------------|\n");
        printf("\t\t|                      1—加法                                 |\n");
        printf("\t\t|                      2—减法                                 |\n");
        printf("\t\t|                      3—乘法                                 |\n");
        printf("\t\t|                      4—除法                                 |\n");
        printf("\t\t|                      0—退出                                 |\n");
        printf("\t\t|*************************************************************|\n");
        printf("\n\n");
        printf("\t\t  请选择运算类型(0-4):");
        scanf("%d",&choice);
        if(choice>=1&&choice<=4)
        {
            printf("\n\t\t  请输入第一个运算数:");
            scanf("%f",&data1);
            printf("\n\t\t  请输入第二个运算数:");
            scanf("%f",&data2);
            printf("\n\t\t  运算结果为:\n");
        }
        switch(choice)
        {
            case 1:printf("\n\t%f+%f=%f\n",data1,data2,data1+data2);break;
```

```
                case 2:printf("\n\t%f-%f=%f\n",data1,data2,data1-data2);break;
                case 3:printf("\n\t%f*%f=%f\n",data1,data2,data1*data2);break;
                case 4:
                if(data2==0)
                printf("\n\t  除数不能为零");
                else
                printf("\n\t  %f*  %f=  %f\n",data1,data2,data1/data2);break;
                case 0:exit(0);          //结束程序的执行,需要包含 stdlib.h 头文件
                default:printf("\n\t  输入选项错误! \n");
            }
            printf("\n\t\t  是否继续计算(输入 y 或 Y 继续,输入其他字符退出)");
            scanf("\n%c",&judge);
        }
    }
```

3.1 顺序结构

结构化程序设计方法普遍采用三种基本程序控制结构，其中顺序结构是最简单的一种，下面将介绍几种 C 语言的语句以及怎样利用它们编写简单的程序。

3.1.1 C 语言的语句

C 语言的语句用来向计算机系统发操作指令，一个语句经编译后产生若干条机器指令。一个实际的程序应当包含若干条语句。C 语言的语句分为以下 5 类。

1. 表达式语句

表达式语句由表达式加上分号“;”组成。

例如：x+y 是表达式，而 x+y; 是语句。

又如：i++是表达式，而 i++; 是语句。

2. 函数调用语句

例如：

```
printf("I Love C!");        /*调用库函数,把字符串 I Love C! 输出 */
```

3. 控制语句

控制语句用于控制程序的流程，以实现程序的各种结构。C 语言有 9 种控制语句，前 8 种将在本章讲述，return 语句将在后续章节讲述。

（1）if…else;

（2）for;

（3）while;

（4）do…while;

（5）continue;

（6）break;

（7）switch;

（8）goto;

（9）return。

4. 复合语句

把多条语句用一对花括号“ {} ”括起来组成的一个语句块称复合语句。例如：

```
{z=x+y;
    c=z/2;
    printf("%f",c);}
```

5. 空语句

只由分号“;”组成的语句称为空语句。空语句是不执行任务的语句。空语句有时用来作流程的转向点，也可用来作为循环体（表示循环体什么也不做）。

3.1.2 赋值语句

赋值语句是由赋值表达式再加上一个分号构成的。

格式：变量=表达式;

赋值语句的功能和特点都与赋值表达式相同，它是程序中最常用的语句。在赋值语句的使用中需要注意以下几点：

（1）C 语言中的赋值号“=”是一个运算符。赋值符“=”右边的表达式可以是一个赋值表达式，可以连续给变量赋值。

例如：a=b=c=d=1;

（2）在定义变量说明中，不允许连续给多个变量赋初值。例如：int a=b=c=d=1;是错误的，应该写成：

int a=1, b=1, c=1, d=1;

（3）注意赋值表达式和赋值语句的区别。赋值表达式可以出现在任何表达式中，赋值语句则不能。

```
例如：a=（b=1）+1;        该语句是正确的
      a=（b=1;）+1;       该语句是错误的，赋值语句不能出现在表达式中
```

3.1.3 数据的输入和输出

C 语言本身不提供输入、输出语句，其操作是由函数来实现的。在 C 标准函数库中提供常用的标准输入/输出函数，有 putchar（输出字符）、getchar（输入字符）、printf（格式输出）、scanf（格式输入）。在使用 C 语言库函数时，要用预编译命令“#include”将有关的“头文件”包括到用户源文件中。例如，在调用标准输入/输出库函数时，文件开头应有以下预编译命令：#include <stdio. h>或#include“stdio. h”。

1. 字符输出函数：putchar（）

putchar（）函数的一般形式为：

```
putchar(c);
```

功能：向终端（一般为显示器）输出一个字符。

说明：c 可以是字符型或整型变量，也可以是一个字符常量或整型常量。

2. 字符输入函数：getchar（）

getchar（）函数的一般形式为：

```
getchar();
```

功能：从键盘上接收输入的一个字符。返回值为一个整数，即输入字符的 ASCII 码。

说明：这是一个不带参数的函数，即圆括号中没有参数，但圆括号不能省略。getchar（）的值可以送给字符变量，也可以送给整型变量。

【例 3.1】输出单个字符。

```
# include <stdio. h>
void main( )
{
    char a=' G' ,b=' O' ,c=' D' ;              /* 定义字符变量 a,b,c 且分别赋初值 */
    putchar( a) ;putchar( b) ;putchar( b) ;putchar( c) ;       /* 输出字符 GOOD */
    putchar( ' \n' ) ;                                          /* 换行 */
    putchar( a) ;putchar( b) ;putchar( c) ;                   /* 输出字符 GOD */
    putchar( ' \n' ) ;
}
```

程序运行结果为：

```
GOOD
GOD
```

【例 3.2】输入单个字符。

```
#include<stdio. h>
main( )
{
    char c;
    c=getchar( ) ;            /*从键盘输入一个字符,保存在字符变量 c 中*/
    putchar( c) ;            /*在屏幕上显示字符变量 c 的值*/
    putchar( ' \n' ) ;
}
```

运行结果为：

```
输入:a↙
输出:a
```

3. 格式输出函数：printf（）

printf（）函数的一般形式为：

```
printf("格式控制",输出表列);
```

功能：按用户指定的格式，把指定的任意类型的数据显示在屏幕上。

说明：

（1）输出格式由格式说明、按原样输出的字符、转义符三部分组成。

（2）格式说明：由“%”和格式字符组成，如%c 和%f 等，作用是将要输出的数据转换为指定格式后输出。

（3）原样输出：普通字符在输出时原样照印，在显示中起提示作用。

（4）转义符为：\n（换行）、\f（换页）或 \t（光标移到下一个制表位）。

在输出时，对不同类型的数据要使用不同的格式字符，常用的格式字符有以下 9 种。

（1）d 格式符：用来输出十进制整数。该格式有以下用法：

1)%d，按十进制整型数据的实际长度输出。

2)%md，m 指出了要输出数据的宽度。若数据位数小于 m，则右对齐，左端补空格；若数据位数大于 m，则按实际位数输出。

例如：

```
a=12;b=12345;
printf("%3d,%3d",a,b);
```

运行结果为：

```
␣␣12,12345
```

3)%ld，输出长整型数据。

例如：

```
long  a=123456;
printf("%ld",a);
```

此处用%d 也可以，因为 VC++ 6.0 中，整型与长整型数据的取值范围相同。

（2）o 格式符：以八进制整数形式输出，输出的数值不带符号，即符号位将作为数值部分输出。

例如：

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

程序运行结果为：

```
-1,37777777777
```

这是因为-1 在内存单元中以补码形式存放（见图 3-1-1）。八进制数为从低位开始，以三位一组划分为一个八进制数，因为 111 为八进制数 7，因此，八进制输出结果为：37777777777。

1	111 1111 1111 1111 1111 1111 1111 1111

图 3-1-1　-1 在内存单元中的存储

（3）x 格式符：以十六进制整数形式输出，输出的数值不带符号，即符号位将作为数值部分输出。

例如：

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

程序运行结果为：

```
ffffffff,37777777777,-1
```

十六进制数为从低位开始，四位一组划分为一个十六进制数，1111 的值为 15，十六进制表示为 f，所以输出为 ffffffff。

（4）u 格式符：用来输出无符号数，以十进制整数形式输出。

一个有符号的 int 型数据可以用%d 格式输出，也可以用%u 格式输出。要注意两种数据类型的取值范围不同，当数值不超过 2 147 483 647 时，用%d 或%u 格式输出结果相同。

【例 3.3】无符号数据的输出。

```
# include <stdio.h>
main()
{
    unsigned int a=65535;
    int b=-2;
    printf("%d,%o,%x,%u\n",a,a,a,a);
    printf("%d,%o,%x,%u\n",b,b,b,b);
}
```

程序运行结果为：

```
65535,177777,ffff,65535
-2,37777777776,fffffffe,4294967294
```

（5）c 格式符：用来输出一个字符。

例如：

```
char c='a';
printf("%c,%d\n",c,c);
```

程序运行结果为：

```
a,97
```

可以看出，一个范围在 0~255 的整数，既可以用%d 格式输出，也可以用%c 格式输出。用%d 格式输出对应的 ASCII 值，用%c 格式输出的是对应的字符。

（6）s 格式符：用来输出一个字符串。该格式有以下用法：

1）%s。

例如：

```
printf("%s","GOOD");
```

程序运行结果为：

```
GOOD
```

2）%±ms：输出占 m 列，如果字符串的实际宽度小于 m，%+ms 右对齐，左端补空格，而%-ms 左对齐，右端补空格；否则，不受 m 限制，输出实际宽度。

3）%±m.ns：输出占 m 列，只取字符串中左端 n 个字符，%+m.ns 右补空格，%-m.ns 左补空格。如果 n>m，则 m 自动取 n 值，保证 n 个字符的正常输出。

【例 3.4】字符串的输出。

```
# include <stdio.h>
main()
{
    printf("%5s,%6.3s,%.3s,%-5.3s\n","chinese","chinese","chinese","chinese");
}
```

程序运行结果为：

```
chinese, ␣␣ ␣␣ ␣␣chi,chi,chi ␣␣ ␣␣
```

（7）f 格式符：以小数形式输出实数。该格式有以下用法：

① %f：不指定字段宽度，整数部分全部输出，并输出六位小数。

② %m.nf：输出数据共占 m 列，其中有 n 位小数。若数的总长度小于 m，则靠右端

对齐，左端补空格。

③ %-m. nf：与%m. nf 用法基本一样，数据输出时靠左端对齐，右端补空格。

例如：

```
float   a=12.34567;
printf("%f,%12f,%12.3f,%.3f,%-12.3f\n",a,a,a,a,a);
```

程序运行结果为：

12.345670, ␣ ␣ ␣ 12.345670, ␣ ␣ ␣ ␣ ␣ ␣ 12.346,12.346,12.346 ␣ ␣ ␣ ␣ ␣ ␣

（8）e 格式符：以指数形式输出实数。该格式有以下用法：

① %e：由系统自动指定给出 6 位小数，指数部分占 5 位（如 e+001），其中“e”占 1 位，“+”占 1 位，指数占 3 位。数值输出时按规范化指数形式（即小数点前只有 1 位非零数字）。

例如：

```
printf("%e",12.345);
```

程序运行结果为：

```
1.234500e+001
```

② %m. ne 和%-m. ne：其中 m 限定了输出宽度，n 限定了输出小数位数，若没有“-”且宽度小于 m，则数据靠右端，左端补空格。否则，数据靠左端，右端补空格。

例如：

```
float   a=12.34567;
printf("%e,%12e,%12.3e,%.3e,%-12.3e\n",a,a,a,a,a);
```

程序运行结果为：

1.234567e+001,1.234567e+001, ␣ ␣ 1.235e+001,1.235e+001,1.235e+001 ␣ ␣

（9）g 格式符：自动选择 f 格式或者 e 格式，输出时自动选择占宽度较小的一种，且不输出无意义的零。

例如：

```
float   a=12.34567;
        printf("%f,%e,%g",a,a,a);
```

程序运行结果为：

```
12.34570,1.234567e+001,12.3457
```

以上介绍了 9 种格式字符，归纳见表 3-1-1。

表 3-1-1　printf（）格式字符

格式字符	说　明
d	以带符号的十进制形式输出整数，正数不输出符号
o	以八进制无符号形式输出整数，不输出前导符 0
x，X	以十六进制无符号形式输出整数，不输出前导符 0x 用 x 时输出十六进制的数码 a~f，以小写形式输出。用 X 时则以大写输出
u	以无符号十进制形式输出整数

续表

格式字符	说　明
c	以字符形式输出，只输出一个字符
s	输出字符串
f	以小数形式输出单、双精度数，隐含输出 6 位小数
e，E	以指数形式输出实数，用 e 时指数以小写表示（如 12. 345e+001），用 E 时指数以大写表示（如 12. 345E+001）
g，G	选用%f 或%e 格式中输出宽度较短的一种格式，不输出无意义的 0。用 G 时，若以指数形式输出，则指数以大写表示

在格式说明中，在% 和上述格式字符之间可以插入以下几种附加符号，见表 3-1-2。

表 3-1-2　printf（）的附加格式说明字符

字符	说　明
l	用于长整型，可加在格式符 d、o、x、u 前面
m（代表一个正整数）	以八进制无符号形式输出整数，不输出前导符 0
n（代表一个正整数）	以十六进制无符号形式输出整数，不输出前导符 0x 用 x 时输出十六进制的数码 a~f，以小写形式输出。用 X 时则以大写输出
-	输出的数字或字符在域内向左靠

当使用 printf（）函数输出时，要说明以下几点：

（1）除了 X、E、G 外，其他格式字符必须用小写字母，如%d 不能写成%D。

（2）如果想输出字符“%”，则应该在“格式控制”字符串中连续用两个% 表示。

例如：

```
printf("%f%%",12.345);
```

程序运行结果为：

```
12.345000%
```

（3）输出表列的问题。

输出表列：给出的各个输出项，要求格式字符和各输出项在数量和类型上应该一一对应。一般形式为：

```
printf(“输出格式”,输出表列);
```

printf（）函数中的“输出表列”既可以是变量名，也可以是表达式。若“输出表列”中有多个输出项，则每个输出项之间用逗号分隔。

4. 格式输入函数 scanf（）

scanf（）函数的一般格式为：

```
scanf(“格式字符”,地址表列);
```

功能：按用户指定的格式从键盘上输入多个相同或不同类型的数据，并将键盘输入的数据转换为指定的格式存放到对应变量的内存地址中。

（1）格式字符。格式字符是由“%”和格式符组成的用双引号括起来的字符串，如%c、%d等。scanf（）函数格式字符见表3-1-3，作用是将输入数据转换为指定格式后，存入由地址表所指的相应变量地址中。

表3-1-3　scanf（）函数格式字符

格式字符	说　明
d	输入十进制整数
o	输入八进制整数
x	输入十六进制整数
c	输入单个字符
s	输入字符串
f	输入浮点数（小数或指数形式）
e	输入浮点数（指数形式）
ld，lo，lx	输入长整型数据
lf，le	输入长浮点型数据（双精度）

（2）地址表列。地址表列部分由变量的地址组成，如果有多个变量，则各变量之间用逗号隔开。

例如：scanf（"%d%c"，&a，&b）；

当使用scanf（）函数时，注意以下几个问题：

（1）在scanf（）函数“格式字符”部分中的每个格式说明符都必须在“地址表列”中有一个变量与之对应，

例如：int　a；float　b；char　c　；

scanf（"%d%f%c"，&a，&b，&c）；

格式说明符必须要与相应的变量的类型一致，即“%d”与&a对应，“%f”与&b对应，“%c”与&c对应。

（2）当格式说明符之间没有任何字符时，在输入数据时，两个数据之间要使用“回车”“Tab”或“空格”键作间隔。

例如：scanf（"%d%d"，&a，&b）；

输入数据：1↙　2↙

或：1Tab键2↙

或：1 └┘ 2↙

【例3.5】输入长方形的两条边长，求其面积并且输出。

```
# include <stdio.h>
main()
{
    float a,b,s;
    printf("请输入长方形两边长:");
```

```
    scanf("%f%f",&a,&b);
    s=a*b;
    printf("长方形面积为:%.2f\n",s);
}
```

运行结果：

```
请输入长方形两边长:1.5 └┘ 2↙
长方形面积为:3.00
```

（3）在格式输入中，不能有转义符'\n'，否则语法错误。如果格式说明符之间包含其他字符，则输入数据时，应输入与这些字符相同的字符作间隔，例如：

scanf（"a=%d，b=%d"，&a，&b）；

输入数据时应：a=3，b=4↙

（4）当用%c 格式输入时，空格、转义符都作为有效字符输入。例如：

scanf（"%c %c %c"，&c1，&c2，&c3）；

若输入：a └┘ b └┘ c↙，则 c1 的值为 a，c2 的值为 └┘，c3 的值为 b。

正确的方法为：abc↙

（5）输入实数时，不能规定精度。例如：

scanf（"%6.2f\n"，&a）；

是错误的。

输入数据时，两个数据之间才要使用空格、Tab 或回车键作间隔。

3.1.4　顺序结构应用

结构化程序设计方法普遍采用三种基本程序控制结构来编写，其中顺序结构是最简单的一种，程序执行顺序自上而下，依次执行完所有语句。

归纳顺序结构的特点如下：

（1）从第一条语句开始顺序执行到最后一条；

（2）每一条语句都执行且只执行一遍，如图 3-1-2 所示。

图 3-1-2　顺序结构

下面通过例子介绍顺序结构程序设计。

【例 3.6】输入梯形的上、下底和高，求其面积并且输出。

由数学知识知梯形的面积公式为：area = $(s1 + s2)\frac{h}{2}$，其中 $s1$、$s2$ 表示梯形上、下

底，h 表示梯形的高。

据此编写程序如下：

```
# include <stdio. h>
main( )
{
    float s1,s2,h,area;
    printf("请输入 s1、s2、h:");
    scanf("%f,%f,%f",&s1,&s2,&h);
    area=(s1+s2) * h/2;
    printf("梯形面积为:%. 2f\n",area);
}
```

运行结果：

```
请输入 s1、s2、h:2 ⌴ 3 ⌴ 4↙
梯形面积为:10. 00
```

3.2 选择结构

选择结构是指通过判断某些特定条件是否满足来决定下一步的执行流程。本节将详细介绍选择结构的实现。

3.2.1 if 语句的三种形式

1. 单分支 if 语句

单分支 if 语句的一般形式为：

```
if(表达式)〈语句〉;
```

首先计算表达式的值。若表达式的值为“真”（非 0），则执行语句；若表达式的值为“假”（0），则直接转到此 if 语句的下一条语句去执行。其流程图如图 3-2-1 所示。

例如：

```
if(a<b)printf("%d",b);
```

如果 a<b 为真，则打印 b 的值，否则执行下面的语句。

【例 3.7】 编写程序，显示用户输入一个整数的绝对值。

由数学知识可知，非负数（正数和 0）的绝对值是它本身，非正数（负数）的绝对值是它的相反数。

据此编写程序如下：

```
# include <stdio. h>
main( )
{
    int   x,y;
    printf("请输入 x:");
    scanf("%d",&x);
    y = x;
    if(x<0)y = -x;/*判断 x 若为负数,则绝对值取其相反数,并保存在变量 y 中*/
```

```
    printf("|x|=%d\n",y);/*输出结果*/
}
```

2. 双分支 if 语句

双分支 if 语句的一般形式为：

```
if(表达式)〈语句 1〉;
else〈语句 2〉;
```

首先判断表达式的值，若表达式的值为“真”（非 0），则执行语句 1；否则，执行语句 2。其流程图如图 3-2-2 所示。

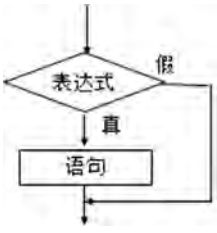


图 3-2-1 单分支 if 语句流程图

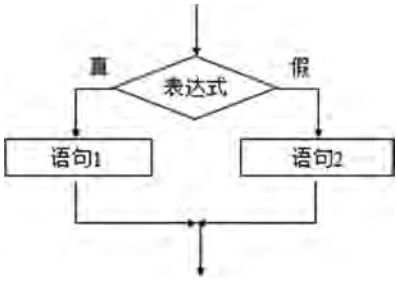


图 3-2-2 双分支 if 语句流程图

例如：

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

如果 a<b 为真，则打印 b 的值，否则打印 a 的值，然后继续执行下面的语句。

【例 3.8】对例题 3.7 还可以这样编写，实现同样的功能。

```
# include <stdio.h>
main()
{
    int   x,y;
    printf("请输入 x:");
    scanf("%d",&x);
    if(x>=0)y = x;  /*判断 x 若为非负数,把 x 的值直接赋给变量 y*/
    else y=-x;     /*否则取 x 的相反数赋给变量 y*/
    printf("|x|=%d\n",y);
}
```

3. 多分支 if 语句

多分支 if 语句的一般形式为：

```
if(表达式 1)〈语句 1〉;
else if(表达式 2)〈语句 2〉;
                …
      else if(表达式 n)〈语句 n〉;
else〈语句 n+1〉;
```

从表达式 1 的值开始进行判断，当出现某个表达式的值为真时，则执行其对应分支的

语句，然后跳出整个 if 语句，执行后续语句。若所有表达式的值都为“假”（为 0），则执行语句 n。其流程图如图 3-2-3 所示。

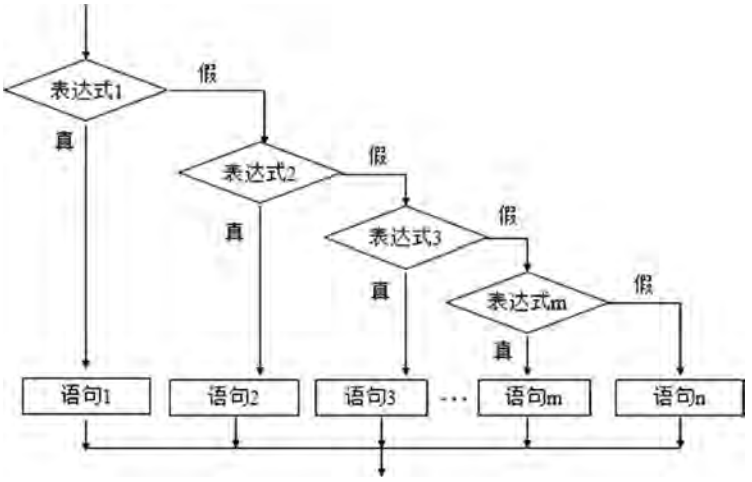


图 3-2-3　多分支 if 语句流程图

【例 3.9】编写程序，输入 x 的值，输出 y 的值，其中 x 与 y 的函数关系如下：

$$y=\begin{cases}0 & (x<0)\\ x & (0<x\leqslant 10)\\ 10 & (10<x\leqslant 20)\\ x+20 & (20<x<40)\end{cases}$$

思路与分析：

按题意，x 的取值范围在 40 以下（不含 40），x 的取值范围对应着计算 y 的表达式，因此要通过讨论 x 的取值范围来确定 y 的值。

如果 $x<0$，$y=0$；

否则如果 $0<x\leqslant 10$，$y=x$；

否则如果 $10<x\leqslant 20$，$y=10$；

否则如果 $20<x<40$，$y=x+20$

程序代码如下。

```
# include <stdio. h>
main( )
{
    int   x,y;
    printf("请输入 x:");
    scanf("%d",&x);
    if(x<0)y = 0;  /*如果 x<0,y=0*/
    else if(x>0&&x<=10)y=x;     /*否则 如果 0<x≤10,y=x*/
    else if(x>10&&x<=20)y=10;  /*否则 如果 10<x≤20,y=10*/
    else if(x>20&&x<40)y=x+20;  /*否则 如果 20<x<40,y=x+20*/
```

```
    else y=-1;                    /*用-1来排除x的非法取值范围外的情况*/
    if(y! =-1)printf("%d\n",y);    /*输出y的值*/
    else printf("error\n");
}
```

3.2.2 if 语句的嵌套

一个 if 语句又包含一个或多个 if 语句，称为 if 语句的嵌套。在 if 语句中可以根据需要，用 if 语句的三种形式进行互相嵌套。一般形式如下：

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

应注意 if 与 else 的配对关系，else 总是与上面最近的尚未配对的 if 配对。例如写成：

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

用户把 else 写在和第一个 if（外层 if）同一列上，希望 else 与第一个 if 对应，但实际上 else 与第二个 if 配对，遵循相距最近原则。因此，if 与 else 的个数最好相同，从内层到外层一一对应，以免出错。如果 if 与 else 的个数不同，可以用花括号来确定配对关系。

例如：

```
if( )
{
    if( )语句 1
}
else 语句 2
```

这时“ {} ”就限定了内嵌 if 语句的范围，确定 else 与第一个 if 配对。

【例 3.10】 编写程序，输入 x 的值，输出 y 的值，其中 x 与 y 的函数关系如下：

$$y=\begin{cases}-1 & (x<0)\\ 0 & (x=0)\\ 1 & (x>0)\end{cases}$$

请读者对比分析下列两种方法。

方法一：

```
# include <stdio.h>
main( )
{
    int  x,y;
    printf("请输入x:");
    scanf("%d",&x);
```

```
    if(x<0)y = -1;
    else if(x==0)y=0;
    else y=1;
    printf("x=%d,y=%d\n",x,y);
}
```

方法二：

```
# include <stdio.h>
main()
{
    int  x,y;
    printf("请输入 x:");
    scanf("%d",&x);
    if(x>=0)
    if(x>0)y=1;
    else y=0;
    else y=-1;
    printf("x=%d,y=%d\n",x,y);
}
```

3.2.3 条件运算符

在 if 语句中，当被判别的表达式的值为“真”或“假”时，都执行一个赋值语句且向同一个变量赋值时，可以用一个条件运算符来处理。

条件运算符的格式：

变量=〈表达式 1〉?〈表达式 2〉:〈表达式 3〉;

执行过程：当表达式 1 的值为“真”时，取表达式 2 的值赋给变量；当表达式 1 的值为“假”时，取表达式 3 的值赋给变量。

用条件运算符可以实现 if 语句的第 2 种形式。

```
if(表达式)  <语句 1>
else <语句 2>
```

例如：

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

用条件运算符可以等价写成：

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

条件运算符是 C 语言中唯一一个三目运算符，其结合性为“从右到左”。

【例 3.11】 任意输入三个整数，输出最小值。

思路与分析：先比较 x 与 y 的值，把较小的保存在变量 min 中，然后再比较此时 min 和 z 的值，返回较小的保存在 min 中。

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

```
    printf("请输入 x,y,z:");
    scanf("%d%d%d",&x,&y,&z);
    min=x>y? y:x;  /*先比较 x 与 y 的值,把较小的保存在变量 min 中*/
    min=min>z? z:min; /*比较此时 min 和 z 的值,返回较小的保存在 min 中*/
    printf("min=%d\n",min);
}
```

3.2.4　switch 语句

switch 语句是一个多分支结构的语句，它所实现的功能与多分支 if 语句很相似，但在大多数情况下，switch 语句表达方式更直观、简单、有效。

1. switch 语句的语法格式

```
switch(<表达式>)
{
    case  <常量表达式 1>:<语句序列 1>;[break;]
    case  <常量表达式 2>:<语句序列 2>;[break;]
    ……
    case  <常量表达式 n>:<语句序列 n>;[break;]
    [default:<语句序列 n+1>;[break;]]
}
```

2. switch 语句的执行过程

switch 语句的执行过程可以用图 3-2-4 表示。

（1）计算 switch 后的表达式的值。

（2）将结果值与 case 后的常量表达式的值比较，如果找到相匹配的 case，程序就执行相应的语句序列，直到遇到 break 语句，switch 语句执行结束；如果找不到匹配的 case，就归结到 default 处，执行它的语句序列，直到遇到 break 语句为止；如果没有 default，则不执行任何操作。

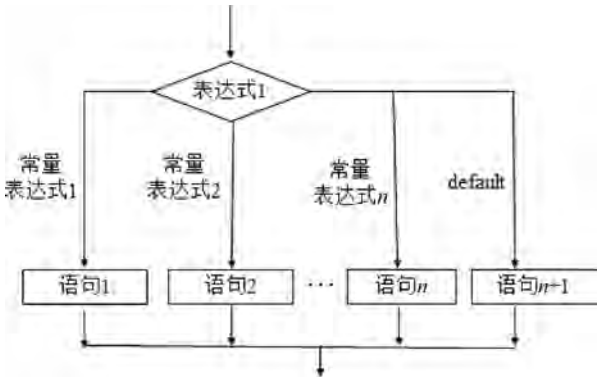


图 3-2-4　switch 语句流程图

【例 3.12】某市不同品牌的出租车 3 千米的起步价和计费分别为：夏利 7 元，3 千米以外 2.1 元/千米；富康 8 元，3 千米外 2.4 元/千米；桑塔纳 9 元，3 千米外 2.7 元/千米。编程：从键盘输入所乘车的车品牌及行车千米数，输出应付车费。

思路与分析：先输入车型（car）和千米数（distance），根据车型和千米数来计算车费。

编程代码如下：

```
#include <stdio.h>
main()
{
    int car;
    double money,distance=0.0;
    printf("请输入出租车类型(1:夏利,2:富康,3:桑塔纳):");
    scanf("%d",&car);
    printf("请输入行车千米数:");
    scanf("%lf",&distance);
    switch(car)
    {
        case 1:  money =(distance<=3)? 7.0:(7.0+2.1*(distance-3));break;
        case 2:  money =(distance<=3)? 8.0:(8.0+2.4*(distance-3));break;
        case 3:  money =(distance<=3)? 9.0:(9.0+2.7*(distance-3));break;
        default:printf("车型输入有误! \n");break;
    }
    printf("应付车费:%.2f\n",money);
}
```

【例 3.13】商场进行打折促销，消费金额（p）越高，折扣（d）越大，具体标准如下：

消费金额	折扣
$p<200$	0%
$200\leqslant p<500$	5%
$500\leqslant p<800$	10%
$800\leqslant p<1000$	15%
$p\geqslant 1000$	20%

要求：从键盘输入消费金额，输出折扣率和实付金额f。

编程代码如下：

```
#include <stdio.h>
main()
{
    int p,d,x;
    float f;
    printf("请输入消费金额:");
    scanf("%d",&p);
    if(p>=0)        /*确保消费金额为非负数*/
    {
        x=p/100;
        if(x>10)x=10;
```

```
        switch(x)
        {
            case 0:
            case 1:d=0;break;  /* 多个 case 可以共用一组执行语句 */
            case 2:
            case 3:
            case 4:d=5;break;
            case 5:
            case 6:
            case 7:d=10;break;
            case 8:
            case 9:d=15;break;
            case 10:d=20;break;
        }
        f=p*(1-d/100.0);
        printf("实付金额:%.2f\n",f);
    }
    else printf("输入有误!");
}
```

3. switch 语句的说明

（1）switch 后面圆括号内表达式的值和 case 后面常量表达式的值，必须是整型或字符型，不允许是浮点型数据。

（2）同一个 switch 语句中的所有 case 后面的常量表达式的值必须互不相同，其中 default 和<语句 n+1>可以省略。

（3）由于 switch 语句中的 case 常量表达式部分只起语句标号的作用，所以在执行完某个 case 后面的语句后，如果没有“break”来结束多分支语句，将自动转到该语句后面的语句去执行，而不再进行条件判断，直到遇到 switch 语句的右花括号为止。

（4）每个 case 的后面既可以是一个语句，也可以是多个语句，当是多个语句的时候，也不需要用花括号括起来。

（5）多个 case 可以共用一组执行语句。

3.3 循环结构

我们要编程解决这样的一个问题：从键盘输入 100 个学生的成绩，求总成绩。

根据前面所学，有两种解决方法：

（1）设 100 个变量，分别输入学生的成绩，然后求和。这种方法浪费内存空间，显然不实际。

（2）设一个变量，每次输入一个学生成绩，累加后再输入下一个学生成绩，如下：

```
scanf("%f",&a);
    s=s+a;
    scanf("%f",&a);
    s=s+a;
    ……
```

这样重复 100 次，然后输出 s 的值。这样写显然非常麻烦。我们注意到程序中的

```
scanf("%f",&a);
s=s+a;
```

两条语句一直是重复的，如果能用一种语句，使这两条语句能自动地重复执行 100 次，就可以简化书写，这就是循环语句。

循环结构可以用较短的语句完成大量的工作，从而大大减少编程的复杂性和工作量。常用的实现循环的语句有 while 语句、do…while 语句和 for 语句，下面一一讲述。

3.3.1 while 语句实现循环

while 语句一般形式如下：

```
循环变量的初始值;
while(循环条件表达式)
循环体语句;
```

循环体语句可以是一条，也可以是多条，多条的时候应用“ {} ”将其括起来，使其成为复合语句。

执行过程：首先计算表达式，如果表达式的值为非零，执行循环体语句，然后返回重新计算表达式，反复执行循环体语句，直到表达式的值为零，则结束循环。如果表达式的值一开始就为 0，则循环体语句一次也不会被执行，其流程图如图 3-3-1 所示。

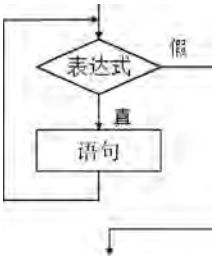


图 3-3-1　while 语句流程图

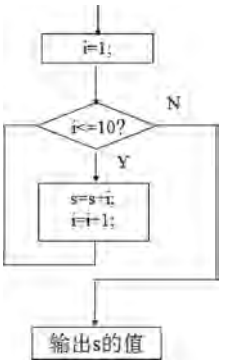


图 3-3-2　例 3.14 流程图

【例 3.14】求 $s=1+2+3+\cdots+10$ 的值，用 while 语句实现。

算法分析（见图 3-3-2）：

（1）定义变量 i 存储加数，定义 s=0 存储累加的和。

（2）输入第一个加数 i。

（3）若 i<=10，执行第（4）步，否则执行第（6）步。

（4）s=s+i。

（5）i++。

（6）输出 s。

参考代码如下：

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

```
{
    int i=1,s=0;
    while(i<=10)                    /*循环条件判断*/
    {
        s=s+i;                      /*花括号内的为循环体*/
        i=i+1;                     /*步长*/
    }
    printf("sum is %d\n",s);  /*输出结果*/
}
```

3.3.2 do…while 语句实现循环

do…while 语句的特点是先执行循环体，然后判断循环条件是否成立。一般形式为：

```
循环变量的初始值;
do
    循环体语句;
while(循环条件表达式);
```

执行过程：先执行一次指定的循环体语句，然后判断循环表达式，当表达式的值为非零（真）时，返回重新执行循环体语句，如此反复，直到表达式的值为 0，结束循环，如图 3-3-3 所示。

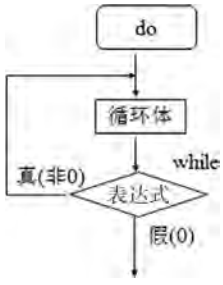


图 3-3-3 do…while 语句流程图

【例 3.15】求 $s=1+2+3+\cdots+10$ 的值，用 do…while 语句实现。

编程代码如下：

```
#include <stdio.h>
main()
{   int i=1,s=0;
    do
    {   s=s+i;                  /*不管循环条件是否成立,循环体至少循环一次*/
        i=i+1;
    } while(i<=10);          /*循环条件判断*/
    printf("sum is %d\n",s);
}
```

do…while 同 while 语句的重要区别：do…while 语句总是先执行一次循环体，然后再求表达式的值，因此，无论表达式的值是 0 还是非 0，循环体至少会执行一次。而对于 while

语句，循环控制在循环体之前，只有当 while 后的表达式为非 0 时，才可能执行循环体，循环体有可能一次都不执行。

3.3.3 for 语句实现循环

for 语句是 C 语言中最常用的一种循环语句，它不仅能用于循环次数已知的情况，还能用于循环次数不确定，而只给出循环结束条件的情况，使用十分灵活方便。

for 语句一般形式为：

```
for(表达式 1;表达式 2;表达式 3)
{
    循环体语句;
}
```

执行过程：

（1）计算表达式 1 的值（初值），然后计算表达式 2 的值（为循环条件）。若表达式 2 的计算结果为“真”（非 0），则执行后续循环语句。循环体语句执行完成后，自动转到表达式 3，计算表达式 3 的值（步长），至此完成一次循环。

（2）计算表达式 2 的值，若结果为“真”（非 0），则继续执行循环体语句。循环体语句执行完成后再转到表达式 3，计算表达式 3 的值（循环步长），至此又完成一次循环。

（3）以上第（2）步反复执行下去，直到计算表达式 2 的值为 0，循环结束，执行 for 语句下面一个语句。

for 语句流程图如图 3-3-4 所示。

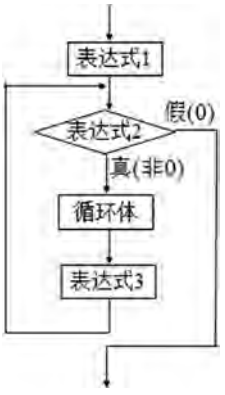


图 3-3-4　for 语句流程图

【例 3.16】求 $s=1+2+3+\cdots+10$ 的值，用 for 语句实现。

编程代码如下：

```
#include <stdio.h>
main()
{   int i,s=0;
    for(i=1;i<=10;i++)   /*先执行 i=1,后判断 i<=10 是否为真*/
    {
        s=s+i;   /*循环体实现累加*/
```

```
    }
    printf("sum is %d\n",s);
}
```

for 语句的一些说明：

（1）for 语句在循环开始时进行条件测试，如果循环体部分是由两条或两条以上语句组成的，则必须用花括号括起来，使其成为一个复合语句。

（2）for 语句中的表达式 1 和表达式 3 既可以是一个简单的表达式，也可以是逗号表达式。

（3）for 语句中的表达式 1、表达式 2、表达式 3 可以省略，但“;”不能省略，如：for（;;），但是会造成死循环，程序无法结束。

3.3.4 循环的嵌套

一个循环体内又包含另一个完整的循环结构，称为循环的嵌套。三种循环结构 while 循环、do…while 循环和 for 循环可以互相嵌套。下面几种形式都是合法的。

```
①while()
{ …
    while()
    { …}
}

②while()
{ …
    do
    {…}while( );
}

③ do
{ …
    do{…}
    while( );
}while( );

④ for(;;)
{…
    do
    {…}while()
}

⑤ for(;;)
{…
    for(;;)
    {…}
}

⑥ for(;;)
{…
    while()
    { …}
}
```

以下两个例子，使用 for 循环嵌套进行编程，读者可自行尝试用其他的嵌套循环去完成。

【例 3.17】 输出九九乘法表。

编程代码如下：

```
#include <stdio.h>
main()
{   int i,j;
    for(i=1;i<10;i++)
    printf("%-8d",i);
    printf("\n--------------------------\n");
    for(i=1;i<10;i++)                    /*外循环,一共循环9次*/
    {
        for(j=1;j<=i;j++)                /*内循环,一共循环i次*/
        printf("%d*%d=%-4d",i,j,i*j);   /*打印结果*/
        printf("\n");
    }
}
```

【例 3.18】输出图 3-3-5 所示的图形。

思路与分析：观察图 3-3-5，可以看成前面 4 行和后 3 行分别遵循不同的分布规律，分别用循环嵌套输出。

图 3-3-5

编程代码如下：

```
#include "stdio.h"
main()
{
    int i,j,k;                          /*定义3个循环变量*/
    for(i=0;i<=3;i++)                   /*控制前4行输出*/
    {for(j=0;j<=i;j++)                  /*控制空格输出*/
    putchar(' ');
    for(k=0;k<=6-2*i;k++)               /*控制星号输出*/
    putchar('*');
    putchar('\n');}
    for(i=0;i<=2;i++)                   /*控制后3行输出*/
    {for(j=0;j<=2-i;j++)
    putchar(' ');
    for(k=0;k<=2*i+2;k++)
    putchar('*');
    putchar('\n');}
}
```

3.3.5 三种循环语句的比较

以上已对三种循环语句一一介绍，下面对它们之间作总结比较：

（1）for 语句功能最强，编写的程序结构简洁、清晰，凡用 while 循环能完成的，用 for 循环都能实现。

（2）不知道确切的执行次数时，使用 do…while 循环。

（3）对于某些语句可能要反复执行多次，也可能一次都不执行的问题，使用 while 循环。

（4）三种循环语句可以相互替代。用 while 和 do…while 循环时，循环变量初始化的操作应在 while 和 do…while 语句之前完成。用 for 语句时，可以放在 for 语句的前面，或在表达式 1 中实现循环变量的初始化。

（5）while 和 for 循环是先判断表达式的值，后执行循环体各语句；而 do…while 循环

是先执行循环体各语句，后判断表达式的值。

3.4 break 语句、continue 语句和 goto 语句

循环结构中还经常应用到一些其他的语句，分别是 break 语句、continue 语句和 goto 语句，下面将一一介绍它们的功能。

3.4.1 break 语句

break 语句的一般形式为：

```
break;
```

功能：在循环语句和 switch 语句中，终止并跳出循环体。

说明：

（1）在多重循环的情况下，使用 break 语句时，仅退出包含 break 语句的那层循环体。

（2）break 语句仅用于开关语句 switch，循环语句 while、do…while 和 for。

【例 3.19】求 100~200 间的全部素数。

```
#include<stdio.h>
#include<math.h>
int main()
{
    int n,k,i,m=0;
    for(n=101;n<=200;n=n+2)/*对每个奇数 n 进行判定*/
    {
        k=sqrt(n);
        for(i=2;i<=k;i++)
        if(n%i==0)break;/*如果 n 被 i 整除,终止内循环,此时 i<k+1*/
        if(i>=k+1)  /*若 i>=k+1,表示 n 未曾被整除,确定为素数*/
        {
            printf("%d ",n);/*输出素数*/
            m=m+1;/*m 用来控制换行,一行内输出 10 个素数*/
        }
        if(m%10==0)printf("\n");/*m 累计到 10 的倍数,换行*/
    }
    printf("\n");
    return 0;
}
```

3.4.2 continue 语句

continue 语句的一般形式为：

```
continue;
```

功能：结束本次循环，跳过循环体中尚未执行的语句，进行下一次是否执行循环体的判断，仅用于循环语句中。

【例 3.20】编程打印出 1~100 之间所有能被 7 整除的数。

```
#include"stdio.h"
void main()
{
    int i;
    for(i=1;i<=100;i++)
    {
        if(i%7!=0)   continue;  /*如果不能被7整除,就提前结束本次循环,不打印结果*/
        else printf("%-3d",i);  /*如果能被7整除,就打印结果*/
    }
    printf("\n");//每一行后换行
}
```

3.4.3 goto 语句

goto 语句的一般形式为：

```
goto〈语句标号〉;
```

功能：goto 语句为无条件转向语句，程序执行到 goto 语句时，无条件地转到〈语句标号〉所指定的语句并执行。

说明：

（1）〈语句标号〉必须用标识符表示，不能用整数作为标号。

（2）goto 语句与 if 语句一起使用，在满足某一条件时，程序跳到标号处执行。

（3）用 goto 语句从循环体中跳到循环体外，这种用法在语法上没有错误，但是不符合结构化原则，一般不宜采用。

【例 3.21】 求 $s=1+2+3+\cdots+10$ 的值。

参考代码：

```
#include <stdio.h>
main()
{   int i=1,s=0;
    loop:if(i<=10)        /*i的值若超过10就结束累加*/
    {
        s=s+i;
        i=i+1;
        goto loop;  /*无条件转向并执行loop所标识的语句*/
    }
    printf("sum is %d\n",s);
}
```

项目二　学生成绩管理系统

项目设置目的

该项目主要使学生加深对基础知识的学习，使学生理解和掌握模块化程序设计的思想和方法，掌握数组和指针的定义和应用，培养学生利用 C 语言进行软件设计的能力。

项目分析

该项目实现了对学生某门课程成绩的统计。主模块应包含菜单显示模块、登录模块、录入信息模块、浏览信息模块、统计总分和平均分模块、统计最高分和最低分模块、统计各分数段人数模块以及退出模块，每个模块都定义为一个功能相对独立的函数。对于本项目中涉及统一类型的大量数据的处理，利用 C 语言中提供的“数组”来存储数据，并通过循环遍历数组的每个元素，来实现计算和查询，使得程序的扩展和维护很容易。

学生成绩管理系统模块结构图如图 4-1 所示。

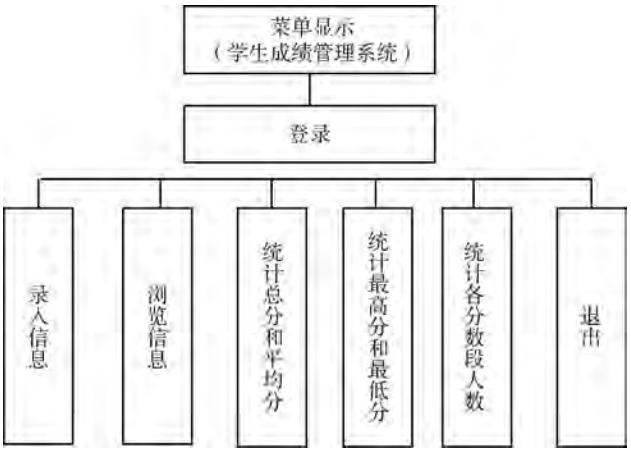


图 4-1　系统模块结构图

系统各模块的功能说明如下：

（1）菜单显示模块，显示整体学生成绩管理系统。

（2）登录模块，实现登录系统时密码的验证。系统初始密码为 abc123。

（3）录入信息模块，实现学生考试成绩的输入。假设输入的第 1 名学生的学号为 1001，第 2 名学生的学号为 1002，依此类推，最多可以输入 30 个学生的成绩，输入-1 即

可终止录入。

（4）浏览信息模块，显示学生成绩。

（5）统计总分和平均分模块，实现某门课程总分和平均分的统计，并显示统计结果。

（6）统计最高分和最低分模块，实现某门课程最高分和最低分的统计，并显示统计结果。

（7）统计各分数段人数模块，实现某门课程各分数段人数的统计。要求将百分制全部转化为优、良、中、及格和不及格 5 个等级，并显示相应的统计结果。

（8）退出模块。

学生成绩管理系统主要是让学生灵活掌握函数、数组和指针等重要知识并提升应用能力。因此该系统实现的功能相对简单，所处理的学生信息也不太全面，在学习结构体和文件内容之后，学生可以自行实现更完善、更实用的学生信息管理系统。

项目分解

任务四　学生成绩管理系统整体框架设计；

任务五　学生成绩管理系统中成绩的统计（1）；

任务六　学生成绩管理系统中成绩的统计（2）。

任务四　学生成绩管理系统整体框架设计

学习目标

- 掌握函数的定义和调用方法；
- 理解和掌握函数的参数传递、函数的重载和函数的默认参数；
- 熟练地使用 C 语句和函数进行程序设计。

一、任务描述

本项目是通过函数来实现模块化程序设计的。项目由多个函数组成，每个函数分别对应各自的功能模块。本任务需要使用结构化编程的思想，实现学生成绩管理系统的整体框架设计。

二、知识要点

本任务涉及结构化程序设计思想和函数的内容，具体内容在理论部分进行介绍。

三、任务分析

根据系统的功能设计出系统操作界面。从系统模块结构图（见图 4-1）可知，本项目的主体模块包括菜单显示模块、登录模块、录入信息模块、浏览信息模块、统计总分和平均分模块、统计各分数段人数模块、退出模块。本任务中将每个模块都定义为一个功能相

对独立的函数，各函数名如下：

（1）菜单显示模块，函数定义为 menu（）。

（2）登录模块，函数定义为 login（）。

（3）录入信息模块，函数定义为 input（int score［］）。

（4）浏览信息模块，函数定义为 output（int score［］，int n）。

（5）统计总分和平均分模块，函数定义为 SumAvg（int score［］，int n）。

（6）统计最高分和最低分模块，函数定义为 MaxMin（int score［］，int n）。

（7）统计各分数段人数模块，函数定义为 grade（int score［］，int n）。

（3）～（7）的函数中都需要学生的成绩信息，因此使用数组 score［］作为形参。

四、源代码参考

```
/*--------------------项目的整体框架实现------------*/
/*==========================预处理命令==========================*/
#include<stdio.h>
#include<stdlib.h>
#include<conio.h>
#include<string.h>
#define MAXSTU 30 //学生人数最大为 30
/*==========================函数原型声明==========================*/
void login();                              //密码验证函数声明
void menu();                               //主菜单函数声明
int  input(int score[]);                   //录入学生成绩函数声明
void output(int score[],int  n);           //浏览学生成绩函数声明
void SumAvg(int score[],int n);            //统计课程总分和平均分函数声明
void MaxMin(int score[],int n);            //统计课程最高分和最低分函数声明
void grade(int score[],int n);             //统计课程各分数段人数函数声明
/*==========================主函数==========================*/
void main()                                //主函数
{
    int score[MAXSTU];                     //定义一维数组,存放学生某门课程的成绩
    int count=0;                           //存放学生实际人数
    int choose;                            //定义整型变量,存放主菜单选择序号
    login();                               //调用密码验证函数
    while(1)
    {
        menu();  //调用显示主菜单函数
        printf("\t\t 请选择主菜单序号(0-5)");
        scanf("%d",&choose);
        switch(choose)
        {
            case 1:count=input(score);  //调用录入学生成绩函数
            break;
```

```
            case 2:output(score,count);  //调用浏览学生成绩函数
            break;
            case 3:SumAvg(score,count);  //调用统计总分和平均分函数
            break;
            case 4:MaxMin(score,count); //调用统计最高分和最低分函数
            break;
            case 5:grade(score,count);  //调用统计各分数段人数函数
            break;
            case 0:return;
            default:printf("\n\n\n 输入无效请重新选择\n");
        }
        printf("\n\n\n 按任意键返回主菜单");
        getch();
    }
}
/*═══════════════════════函数定义部分═══════════════════════*/
void login()                                    //登录函数
{
    printf("请输入密码:\n");
    getch();
}
void menu()                                     //主菜单函数
{
    system("cls");
    printf("\n\n");
    printf("\t\t ***************************************************************\n");
    printf("\t\t                    学生成绩管理系统                           \n");
    printf("\t\t ***************************************************************\n");
    printf("\t\t                    1—录入学生成绩                             \n");
    printf("\t\t                    2—显示学生成绩                             \n");
    printf("\t\t                    3—统计总分和平均分                         \n");
    printf("\t\t                    4—统计最高分和最低分                       \n");
    printf("\t\t                    5—统计各分数段人数                         \n");
    printf("\t\t                    0—退出                                     \n");
    printf("\t\t ***************************************************************\n");
}
int input(int score[])  //输入学生成绩函数
{
    printf("输入学生成绩\n");
    return 0;
}
void output(int score[],int n)
```

```
{
    printf("显示学生成绩\n");
}
void SumAvg(int score[],int n)
{
    printf("统计总分和平均分\n");
}
void MaxMin(int score[],int n)
{
    printf("统计最高分和最低分\n");
}
void grade(int score[],int n)
{
    printf("统计各分数段人数\n");
}
```

由于还未学习数组和字符串的内容，因此本任务只实现了整体框架的设计和主菜单函数 menu（）的整体代码。其他函数的功能在任务五和任务六中实现。

在本任务中，主函数位于自定义函数前面，这种情况下需要先声明自定义函数，也可以将主函数放置在自定义函数之后或者两个自定义函数之间。

4.1 函数的分类

C 语言是通过函数来实现模块化程序设计的。因此，一般的 C 语言应用程序是由多个函数组成的，每个函数分别对应各自的功能模块。

模块化程序设计思想是把一个大的程序按功能进行分解，由于分解后的各模块较小，所以容易实现，也容易调试。

划分模块的基本原则是，各模块都要易于理解，功能尽量单一，模块间的联系尽量少。满足这些要求的模块具有以下优点：

（1）模块间的接口关系简单，这种程序可读性和可理解性较强。

（2）需要修改某一功能时，只涉及一个模块，不会影响到其他模块。

（3）脱离程序的上、下文也能单独验证一个模块的正确性。

（4）扩充或建立新系统时，可充分利用已有的模块。

例如，学生成绩管理系统的开发，先给出项目的整体框架设计，然后具体实现每个功能模块。这种设计思想就像搭积木，单个的积木就是一个模块，它们的功能单一，便于开发与维护。

C 语言函数从不同的角度可以分为不同的类型：

（1）从用户使用的角度分类。

1）库函数。由 C 语言系统提供，用户无须定义，也不必在程序中作类型说明，只需在程序前包含该函数原型的头文件即可在程序中直接调用。在前面的例题中反复用到的 printf（）、scanf（）、getchar（）、putchar（）、gets（）、puts（）、strcpy（）等函数均属此类。

2）用户自定义函数。对于用户自定义函数，不但要在程序中定义函数本身，而且在主调函数模块中还必须对该被调函数进行类型说明，然后才能使用。

（2）从函数完成的任务分类。

1）有返回值函数。该类函数运行结束时，将计算结果返回到主调函数。

2）无返回值函数。该类函数运行结束时，没有数据返回，它只是完成某一种操作。

（3）从函数的表示形式分类。

1）无参函数。主调函数没有将数据传递给被调函数，一般用来完成某一种操作，无参函数可以带回或不带回函数值到主调函数。

2）有参函数。调用该类函数时，在主调函数和被调函数之间有数据传递。主调函数可以将数据传递给被调函数使用，被调函数的计算结果也可以带回主调函数使用。

4.2 函数的定义、调用和声明

函数是用来完成一定功能的。所谓函数名，就是给该功能起一个名字。如果该功能是用来实现数学运算的，就是数学函数。在 C 语言中，往往把程序需要实现的一些功能分别编写为若干个函数，然后把它们组合成一个完整的程序。函数是具有独立功能的一个程序，它可以反复使用，也可以作为一条语句在程序的任何地方使用。

一个 C 语言程序由一个主函数（main）和若干个其他函数构成。主函数可以调用其他函数，其他函数可以相互调用，但不能调用主函数。这些函数可以是库函数，也可以是用户自定义函数。

4.2.1 函数的定义

一个函数必须定义后才能使用。所谓定义函数，就是编写完成函数功能的程序块。C 语言函数由函数头与函数体两部分组成，其一般形式如下：

```
[<返回类型>] <函数名>([形式参数列表])  //函数头
{
    <函数体>                          //函数体
}
```

函数体可以包含若干个变量和对象的定义，以及各种语句序列。

1. 函数头

函数头的组成形式如下：

```
[<返回类型>] <函数名>([形式参数列表])
```

（1）返回类型。

返回类型规定函数返回值的类型。对有返回值的函数，一般通过函数调用得到一个确定值，这个值就是函数返回值（简称函数值）。如 int fnsum（int a，int b）将返回一个 int 类型的值。

对无返回值的函数，函数名前应加上 void 类型，不带返回语句，表示该函数执行后不返回函数值。函数类型标识符如果省略，默认为 int。

（2）函数名。函数名应是一个有效的标识符。为了增加程序的可读性，一般取有助于记忆、与其功能相关的标识符作为函数名。

（3）形参列表。在定义函数时，函数名后面的括号中的变量称为形式参数，简称形参。写在函数名后圆括号中的一组变量名，如果形参列表有多个参数，则它们之间要用“,”隔开。例如 int sum（int a，int b）。

函数头部的参数列表可以为空，即参数列表不包括任何参数，此类函数称为无参函数。

注意：

（1）定义无参函数时，函数名后面的圆括号不能省略。

（2）定义有参函数时，如果包含多个参数，要单独定义每个参数的数据类型。例如：

int sum（int a，int b）形式正确

int sum（int a，b）形式错误

2. 函数体

用一对花括号 { } 括起来的部分称函数体，它由若干条语句组成，描述函数实现一个功能的过程。一般包括变量的定义和声明部分、语句序列部分。

函数体可以为空，称为空函数。调用空函数时，不做任何操作，但是可以声明在此要调用一个函数，以后扩展函数功能时再补充。

【例 4.1】实现返回两个整数最大值的函数。

```
int max(int a,int b)
{
    if(a>b)
    return a;  //a
    else
    return b;     //b
}
```

当函数的形式参数 a 大于 b 时，执行 a 行语句 return a；当 a 小于 b 时，执行 b 行语句 return b。

注意：

（1）不允许在一个函数体中再定义另一个函数，即函数不能嵌套定义。

（2）不同函数定义放置位置没有关系，可以定义在 main 函数之前，也可以定义在 main 函数之后。

函数体定义的变量只能用在本函数的内部，称为该函数的局部变量。参数列表中定义的变量用于接受实际参数的值，也只能在该函数内部使用。因此，形式参数也属于局部变量。关于局部变量将在后面的章节进行介绍。

4.2.2 函数的值

根据函数的不同功能，将 C 语言的函数分为两类，一类函数用于计算一个值，称为具有返回值的函数；另一类函数仅仅是为了实现一个过程，而不是为了得到一个值，称为无返回值的函数。

return 语句的一般形式为：

```
return 表达式;  或  return(表达式);  或  return;
```

说明：

（1）当函数执行到 return 语句时，返回到它的主调函数的调用位置，并带回返回值。

（2）return 后的表达式可以是常量、变量或表达式。

（3）表达式的类型若和函数定义中的返回值类型不相同，则系统自动转换为定义的类型；若无法转换，则赋值不兼容。

（4）若函数定义为 void 类型，则不能用 return 带回返回值。函数最后一个“}”起返回作用。

（5）函数中可以有多条 return 语句，但只执行其中一条，或都不执行。

（6）如果不需要返回值，则使用无表达式的 return 语句。

1. 无返回值的函数

在 C 语言程序中，每一个运算对象都有数据类型，函数的返回值也不例外。如果函数仅仅是为了表示一个过程，不需要返回值，应该将函数返回值的数据类型设置为空类型(void)。否则，定义函数时，应该定义函数返回值的数据类型。

【例 4.2】实现求一个整数的绝对值的函数。

```
#include <stdio.h>
void fnabs(int a)   //a 为形参,无返回值
{
    if(a>=0)
        printf("x 的绝对值为%d\n",a);
    else
        printf("x 的绝对值为%d\n",-a);
}
void main()
{
    int x;
    printf("输入一个整数 x:");
    scanf("%d",&x);
    fnabs(x);                  //x 为实参
}
```

无返回值的函数只能作为单独语句，而不能作为算术或赋值等运算符的操作数进行相应的算术或赋值运算。

2. 有返回值的函数

该类函数在结束时，将计算结果返回到主调函数。

返回的作用如下：

（1）将流程从当前函数返回其上级（调用函数）。

（2）撤销函数调用时为各参数及变量分配的内存空间。

（3）向调用函数返回最多一个值。

一般来说，函数的返回由返回语句来实现。

注意：

（1）定义函数的头部时，如果不指明函数返回值的数据类型，编译器默认该函数的返回值数据类型为 int。

（2）有返回值的函数的计算结果，只有通过返回语句，才能向函数的使用者返回指定的计算结果。

【例 4.3】实现返回一个整数的绝对值的函数。

```
#include <stdio.h>
int fnabs(int a)//a 为形参
{
    if(a>=0)
        return a;
    else
        return -a;
}
void main()
{
    int x;
    printf("输入一个整数 x:");
    scanf("%d",&x);
    printf("x 的绝对值为%d\n",fnabs(x));  //x 为实参
}
```

程序分析：

程序运行时，实参将 x 的值传给形参 a，在函数 fnabs（）中，将绝对值用 return 语句返回给主调函数，并将该值打印输出。

4.2.3 函数的声明和调用

1. 函数的原型与声明

在 C 语言中，当函数定义在前、调用在后时，调用不必声明；如果一个函数定义在后、调用在前，为了遵守“名字必须先说明后使用”的原则，必须先对函数进行原型说明，也就是函数声明。函数声明的一般格式如下：

```
<存储类型符/返回值类型符> 函数名(形式参数列表);
```

如例 4.4 中：

```
void fnpower(float x,int n);
```

说明了一个函数，也称提供了一个函数的原型，因为它反映了函数的类型、函数名以及形参的个数、类型和顺序，但是形参的名字是不重要的，可以不写。fnpower（）函数说明还可以写成：

```
void fnpower(float,int);
```

函数说明通常出现在程序的开头，第一个函数定义之前，也可以放在主调函数的开头。

2. 函数的调用

使用已经定义的函数的过程，称为函数的调用。函数的调用方式可以分为一般调用、嵌套调用和递归调用三种。

C 语言中，一般函数调用可以分为无返回值函数的调用和有返回值函数的调用两种形式。

（1）无返回值函数的调用。

只要把函数作为一条语句处理，无返回值函数的调用格式为：

```
<函数名>([<实参列表>]);
```

这时不需要函数返回值，只要求函数完成一定的功能。

（2）有返回值函数的调用。

把函数返回值赋给调用函数中的某个变量，一般形式为：

```
<变量> = <函数名>([ <实参列表> ]);
```

有返回值调用时实参的个数、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。

【例 4.4】编写函数，输出 x 的 n 次幂。

```
#include <stdio.h>
void main()
{
    void fnpower(float x,int n);//声明函数
    float x;
    int n;
    printf("输入底数 x,指数 n:");
    scanf("%f,%d",&x,&n);
    fnpower(x,n);
}
void fnpower(float x,int n)   //定义函数
{
    int i;
    float p=1.0f;
    for(i=1;i<=n;i++)
        p*=x;
    printf("%f 的%d 次幂是:%f",x,n,p);
}
```

3. 调用库函数

调用库函数时，通常在文件开头用#include 命令将调用有关库函数时所需用的信息“包含”到本文件中来。如：标准输入/输出库函数用#include <stdio.h>，数学库函数用#include <math.h>。常用库函数见附录一。

根据函数在程序中出现的位置，大致有三种调用方式。

（1）把函数调用作为一个语句。如：

```
fnpower(x,n);
```

这种调用不返回函数值，只要求函数完成一定的操作，如例 4.4 所示。

（2）函数出现在一个表达式中，这时要求函数带回一个确定的值以参加表达式的运算。如：

```
x=2*max(a,b);
```

（3）一个函数作为另一个函数的实参。

4.2.4　函数的嵌套调用和递归调用

上节介绍的函数调用方式是一个函数调用另一个函数，这种调用属于一般调用。C 语言还允许函数的嵌套调用和递归调用。

1. 函数的嵌套调用

C 语言定义的函数都是互相独立的，函数间不能嵌套定义（嵌套定义是指定义一个函数时，其函数体内包含另一个函数的完整定义），但可以嵌套调用，也就是说在调用一个函数的过程中，该函数又调用另一函数。其调用关系如图 4-2-1 所示。

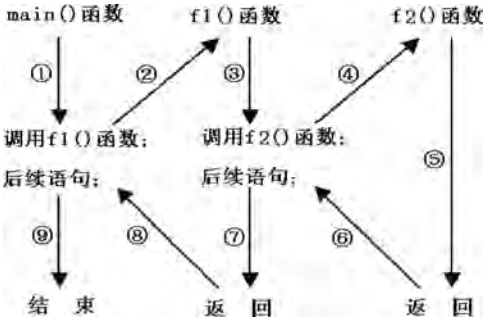


图 4-2-1　函数的嵌套调用

图 4-2-1 表示了两层嵌套调用的情形，其执行过程是：首先执行 main（）函数，当遇到调用 f1（）函数的语句时，即转去执行 f1（）函数；在 f1（）函数的执行过程中，当遇到调用 f2（）函数的语句时，又转去执行 f2（）函数；f2（）函数执行完毕返回 f1（）函数的调用点继续执行，f1（）函数执行完毕返回 main（）函数的调用点继续执行，直到整个程序结束。

【例 4.5】编写函数，输出 3 个数的最大值，用函数嵌套调用实现。

```
#include <stdio.h>
void main()
{
    int max(int x,int y);
    int a,b,c,m;
    scanf("%d%d%d",&a,&b,&c);
    m=max(a,max(b,c));
    printf("最大值是:%d\n",m);
}
int max(int x,int y)
{
    if(x>y)
```

```
        return x;
    else
        return y;
}
```

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本身，如图4-2-2所示。

图 4-2-2　函数的递归调用

一个函数在其函数体内直接地或间接地调用自己本身，称为函数的递归调用。显然，递归调用是嵌套调用的特例。

（1）函数递归调用的条件。

利用函数递归调用解决问题，必须具备如下两个条件：

1）原问题求解，能转化为一个与原问题相似的较小的问题求解。

2）必须有一个明确的递归结束条件，称为递归出口。

（2）函数递归调用的执行过程。

函数的执行过程由递推和回归两个阶段组成。

① 递推阶段：将原问题不断地分解为新的子问题，逐步从未知的方向向已知方向推测，最终达到已知的结束条件，即递归结束条件，这时递推阶段结束。

② 回归阶段：从已知的条件出发，按照递推的逆过程，逐一求值返回，直到返回到递推的开始处，结束回归阶段，完成递归调用。

递归算法编程的要点：第一是找到相似性，把原始的问题转化为相似的小问题，递归调用。第二是设计出口，即递归的终结条件。

递归函数的典型例子是求阶乘。下面通过求阶乘的函数，详细分析递归过程，如图4-2-3所示。

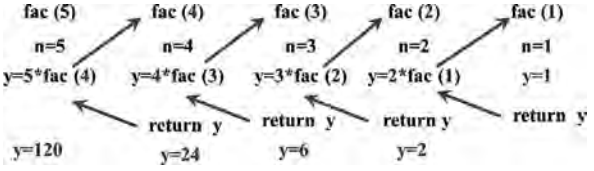


图 4-2-3　$n!$ 递推和回归的过程

（1）递推阶段：阶乘的计算公式为：$n!=n\times(n-1)\times\cdots\times2\times1$。即 $n!$ 等于 $n\times(n-$

1)!，而（n-1)！又等于（n-1）×（n-2)！依此类推，一直推到1！=1，0！=1。

（2）回归阶段：0和1就是这个递归函数的出口。

【例4.6】编写函数，求n!，用函数递归调用实现。

```
#include <stdio.h>
void main()
{
    int factor(int);          //函数声明,函数原型
    int n,i;
    printf("请输入一个整数:");
    scanf("%d",&n);           /*输入一个数*/
    i=factor(n);              /*调用递归函数*/
    printf("%d! =%d\n",n,i);  /*打印返回的值*/
}
int factor(int n)             /*递归函数*/
{
    if(n<0)
        printf("n不可以小于0!");
    else if(n==0||n==1)
        return 1;
    else
        return(n*factor(n-1));/*递归调用*/
}
```

【例4.7】5个人坐在一起，问第5个人多少岁？他说比第4个人大2岁。问第4个人的岁数，他说比第3个人大2岁。问第3个人，他说比第2个人大2岁。问第2个人，他说比第1个人大2岁。最后问第1个人，他说是10岁。请问第5个人的岁数。

```
#include <stdio.h>
int main()
{
    int age(int n);
    int n=5;
    printf("第5个人的年龄为:%d\n",age(n));
    return 0;
}
int age(int n)
{
    if(n==1)
        return 10;
    else
        return 2+age(n-1);
}
```

4.3 函数参数的传递

在调用函数时，主调函数与被调函数之间大都有数据传递关系。主调函数向被调函数传递数据是通过函数的参数进行的，而被调函数向主调函数传递数据一般是利用 return 语句实现的。在使用函数的参数传递数据时，可以采用两种方式，即按值传递和地址传递。从本质上讲，C 语言中只有传值方式，因为地址也是一种值，传址实际上是传值方式的一个特例，只是为了讲述方便，将它们分开讨论。本节只介绍按值传递。地址传递将在后面章节中进行详细介绍。

函数调用时，主调函数的参数称为实参，被调函数的参数称为形参。所谓按值传递，是指当一个函数被调用时，根据实参和形参的对应关系将实参一一传递给形参，供函数执行时使用。函数本身不对实参进行操作，也就是说，即使形参的值在函数中发生了变化，实参的值也不会受影响。这样的参数也称为传值参数。

使用传值方式在函数间传递数据时应注意以下几点：

（1）实参和形参的类型、个数和顺序都必须保持一致，实参可以是常量、变量、表达式或数组元素，但必须有确定的值，以便把这些值传送给形参。因此，应预先用赋值、传输等方法使实参获得确定的值。

（2）形参变量只有在函数被调用时才分配存储单元，函数调用结束后，即释放所分配的存储单元。因此，形参变量只有在该函数内有效，函数调用结束返回主调函数后，就不能再使用该形参变量。

（3）实参对形参的数据传递是单向的值传递，即只能把实参的值传递给形参，而不能把形参的值反向传递给实参，因此形参的改变并不影响实参。

（4）形参只作用于被调函数，可以在别的函数中使用相同的变量名。

【例 4.8】值传递在函数之间传递数据举例。

```
#include <stdio.h>
void main()
{
    void swap(int,int);
    int a,b;
    printf("请输入两个整数\n");
    scanf("%d%d",&a,&b);
    swap(a,b);                    //调用 swap()函数
    printf("a=%d,b=%d\n",a,b);
}
void swap(int x,int y)   //定义 swap()函数
{
    int t;
    t=x;                          //交换 x,y 的值
    x=y;
    y=t;
```

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

输入 2、3，程序运行结果为：

```
x=3,y=2
a=2,b=3
```

在该程序中，函数间的数据传递采用传值方式，当执行到 main（）函数中的函数调用语句“swap（a，b);”时，给 swap（）函数中的两个形参 x 和 y 分配存储空间，并将实参 a、b 的值 2 和 3 分别传递给 x 和 y。此时数据传递如图 4-3-1 所示。在执行 swap（）函数时，确实交换了 x 和 y 的值，但当函数调用结束返回主函数时，形参 x 和 y 所占的存储空间被释放，形参值的改变并不能影响实参。因此，主函数中 a 和 b 的值维持不变，并未实现两数的交换。返回 main（）函数时，实参和形参的情况如图 4-3-2 所示，其中虚框表示形参所占内存空间已被释放。

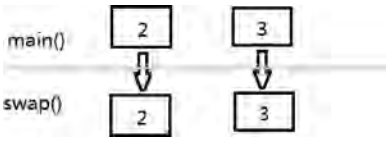


图 4-3-1　值传递前

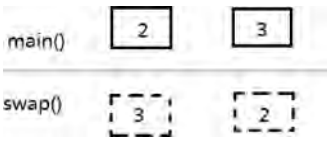


图 4-3-2　值传递后

4.4　变量的作用域

C 语言中所有的变量都有自己的作用域。变量说明的位置不同，其作用域也不同。所谓变量的作用域就是指变量能被有效引用的范围。从“变量的作用域”角度来分，C 语言将变量分为局部变量和全局变量。

1. 局部变量

一般来说，在一个函数内部声明的或在一个块中定义的变量是局部变量，其作用域只在本函数范围内，即局部变量只能在定义它的函数体内部使用，而不能在其他函数中使用。例如：

```
int  fnFunction(int x,int y)
{
    int  i,j;        //i,j 均在函数内定义,属于局部变量,在 main()函数中不能访问
      …  //省略
}
void main()
{  int  a,b;              //a,b 是主函数中的局部变量,在 fnFunction()函数中不能访问
        …                 //省略
    fnFunction(a,b);
    …                     //省略
}
```

2. 全局变量

在函数外部定义的变量称为全局变量，可被作用域内的所有函数直接引用。

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。全局变量的作用域示列如下：

```
int   p=1,  q=5; //全局变量                          }
float   f1(int   a)                                  }
{   int   b,c;  //局部变量        }a,b,c 有效       }
    .....                         }                  }
}                                                    }p,q 有效
char   c1,c2;                                        }
main( )                                              }
{   int   m,n;                    }c1,c2 有效        }
    .....          }m,n 有效      }                  }
}                                                    }
```

全局变量可加强函数模块之间的数据联系，但又使各函数依赖这些全局变量，因而使得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，因此不要使用外部变量。

在同一源文件中，外部变量与局部变量可以同名，因为它们的作用范围不同，所以它们相互没有影响。

【例 4.9】 全局变量和局部变量的作用域。

```
#include <stdio. h>
int   a=3,b=5;              //定义并初始化全局变量
int max(int   a,int   b)
{
    int c;                  //max( )中的局部变量
    c=a>b? a:b;
    return   c;
}
void main( )
{
    int   a=8;              //main( )中的局部变量
    printf("最大值为:%d\n",max(a,b));
}
```

本例中 a 和 b 在程序开始就声明，是全局变量。当 main（）函数调用 max（a，b）时，a 的值是使用 main（）中的局部变量初始化的值，而 b 是使用全局变量初始化的值。因此将（8，5）传递给 max（）函数，作为形参，执行程序，如图 4-4-1 所示。然后将 c 返回给 main（）函数。

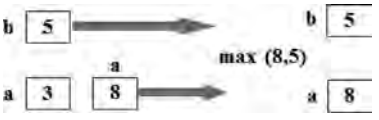


图 4-4-1　例 4.10 中变量的作用域

4.5 编译预处理

C 语言程序与其他高级语言程序的一个重要区别是可用预处理命令和具有预处理的功能。运用预处理命令可以改进程序设计环境，提高编程效率。

这些预处理命令是由 ANSI C 统一规定的，但它不是 C 语言本身的组成部分，不能直接对它们进行编译。在对程序进行通常的编译之前，必须先对程序中这些特殊的命令进行“预处理”，即根据预处理命令对程序作相应的处理。预处理器对宏进行替换，并将所包含的头文件整体插入源文件中，为后面要进行的编译做好准备。

C 语言提供的预处理功能主要有以下三种：宏定义、文件包含、条件编译，分别用宏定义命令、文件包含命令、条件编译命令来实现。为了与一般 C 语句相区别，这些命令以符号“ #”开头。

4.5.1 宏定义

在 C 语言程序中，用一个标识符来表示一个字符串，称为“宏”，被定义为“宏”的标识符则称为“宏名”。宏定义可分为两种形式：不带参数的宏定义和带参数的宏定义。

1. 不带参数的宏定义

不带参数的宏定义的一般形式为：

```
#define  符号常量名  字符串
```

其中符号常量名称为宏名，习惯用大写字母表示，符号常量名和所对应的字符串之间用空格隔开。在定义后，凡是出现符号常量名的地方，经编译预处理后都被替换为它对应的字符串，即“宏展开”（宏代换）。

【例 4.10】不带参数的宏定义。

```
#include<stdio.h>
#define M  (y*y+3*y)
void main()
{
    int s,y;
    printf("Input a number:");
    scanf("%d",&y);
    s=3*M+4*M+y*M;
    printf("s=%d\n",s);
}
```

2. 带参数的宏定义

带参数的宏定义的一般形式为：

```
#define 宏名(参数表)  字符串
```

其中，字符串应包含在参数表中所指定的参数。如例 4.11 所示。

【例 4.11】带参数的宏定义。

```
#include<stdio.h>
#define  SQ(y)  ((y)*(y))
void main()
```

```
{
    int  i =1;
    while(i<=5)
        printf("%d\t",SQ(i++));
}
```

如果善于利用宏定义，可以实现程序的简化，如事先将程序中的输出格式定义好，以减少在输出语句中每次都要写出具体的输出格式的麻烦。

4.5.2 文件包含命令

文件包含处理是指一个源文件可以将另外一个源文件的全部内容包含进来。C 语言提供了#include 命令用来实现文件包含的操作。一般形式为：

```
#include "文件名"
```

或

```
#include <文件名>
```

两种形式的区别在于：使用尖括号表示在系统头文件目录中查找（由用户在设置编程环境时设置），而不在源文件目录中查找；使用双引号则表示首先在当前的源文件目录中查找，找不到再到系统头文件目录中查找。

文件包含在程序设计中非常重要。一个大的程序通常分为多个模块，由多个程序员分别编程。有些共用的数据（如符号常量和数据结构）或函数可组成若干个文件，凡是要使用其中数据或调用其中函数的程序员，只要使用文件包含命令将所需文件包含进来即可，不必再次定义，从而减少重复工作。

【例 4.12】 文件包含应用举例。

```
#include <stdio.h>
#include "4-14.c"
void main()
{
    int  a,b;
    printf("请输入两个整数:");
    scanf("%d%d",&a,&b);
    printf("最大值为:%d\n",max(a,b));
}
```

以下是 4-14.c 的代码：

```
int max(int  a,int  b)
{
    int c;              //max()中的局部变量
    c=a>b? a:b;
    return  c;
}
```

4.5.3 条件编译

通过某些条件，控制源程序中的某段源代码是否参加编译，这就是条件编译的功能。

一般来说，所有源文件中的代码都应参加编译，但有时候希望某部分代码不参加编译，应用条件编译可达到这一目的。

条件编译的基本形式有三种：

（1）#if 表达式。

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

其功能为：如果“表达式”的值为真（非 0），则编译“程序段 1”，否则编译“程序段 2”。如果“表达式”的值为假（0）时，直接跳过#endif。表达式在编译时求值，只能是已经定义的标识符和常数，不能使用任何变量。表达式不要求用圆括号括起来。

（2）#ifdef 标识符。

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

其功能为：如果#ifdef 后面的“标识符”已经被#define 命令定义过，则编译“程序段 1”；否则编译“程序段 2”。如果没有#else 部分，则当标识符未定义时直接跳过#endif。

（3）#ifndef 标识符。

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

其功能为：如果#ifndef 后面的“标识符”没有被#define 定义过，则编译“程序段 1”，否则编译“程序段 2”。如果没有#else 部分，则当标识符已定义，直接跳过#endif，该形式与第（2）种形式的功能正好相反。

任务五　学生成绩管理系统中成绩的统计（1）

学习目标

- 掌握一维数组和二维数组的定义、初始化和使用方法；
- 理解字符数组和字符串的区别，掌握它们的使用方法。

一、任务描述

在任务四中，函数的函数体均是用一条输出语句，本任务中将使用数组来实现各函数

的功能。

二、知识要点

本任务中主要使用一维数组来存储成绩，并进行传递。

三、任务分析

通过前面的学习，使用基本数据类型（整型、实型、字符型），可以实现数据的存储和处理。但是在实际问题中面对大量的数据，如果仍用基本数据类型来进行处理则很不方便。因此本任务中定义一个整型一维数组 score［］来存放学生成绩，并且定义一个符号常量 MAXSTU，用于定义数组的长度。

经分析，自定义函数基本都用到数组 score［］。数组的访问有以下两种方法：

（1）将该数组定义为全局变量，每个函数均可直接访问数组。

（2）将该数组定义为局部变量，利用实参和形参的数据传递，实现对学生成绩数据的访问。

若采用第一种方法，则函数之间的关联性较强，数据的安全性很难保证。因此，本任务采用第二种方法，在主函数中将整型数组 score［］定义为一个局部变量，数组元素的下标对应学生的学号；再定义一个局部变量 stu_count，存放学生的实际人数（即数组的实际长度）。在进行函数调用时，将数组 stu_score 和数组实际长度 stu_count 作为实参，传递给其他函数的形参，从而实现对学生成绩数据的访问。

四、源代码参考

```
/*═══════════════════函数定义部分═══════════════════*/
void login()                                    //登录函数
{
    char pwd[10]="abc123";
    char ch[10];
    int re;
    printf("请输入密码:\n");
    gets(ch);
    re=strcmp(ch,pwd);
    if(re==0)
        puts("密码正确,登录成功");
    else
    {
        puts("密码不正确,请重新输入:");
        login();
    }
}
void menu()                                     //主菜单函数
{
    system("cls");
    printf("\n\n");
```

```
    printf("\t\t ************************************************************\n");
    printf("\t\t                  学生成绩管理系统                          \n");
    printf("\t\t ************************************************************\n");
    printf("\t\t                  1—录入学生成绩                            \n");
    printf("\t\t                  2—显示学生成绩                            \n");
    printf("\t\t                  3—统计总分和平均分                        \n");
    printf("\t\t                  4—统计最高分和最低分                      \n");
    printf("\t\t                  5—统计各分数段人数                        \n");
    printf("\t\t                  0—退出                                    \n");
    printf("\t\t ************************************************************\n");
}
int input(int score[])  //输入学生成绩函数
{
    int i;
    printf("\n 输入学生成绩(输入-1 退出)\n");
    for(i=0;i<MAXSTU;i++)
    {
        printf("\t 第%d 个学生的成绩为:",i+1);
        scanf("%d",&score[i]);
        if(score[i]==-1)
            break;
    }
    return i;           //返回学生的实际人数
}
void output(int score[],int n)  //显示学生成绩函数
{
    int i;
    printf("\n\n 学生成绩:");
    printf("\n 学号\t\t 成绩");
    for(i=0;i<n;i++)
    {
        printf("\n%d\t\t%d",1001+i,score[i]);
    }
}
void SumAvg(int score[],int n)     //统计总分和平均分
{
    int i,sum=0;
    float ave=0;
    for(i=0;i<n;i++)
    {
        sum+=score[i];
    }
```

```
    ave=(float)sum/n;
    printf("\n 总分为%d,平均分为%.2f\n",sum,ave);
}
void MaxMin(int score[],int n)
{
    int i,max=0,min=0;
    max=score[0];
    min=score[0];
    for(i=0;i<n;i++)
    {
        if(score[i]>max)
            max=score[i];
        if(score[i]<min)
            min=score[i];
    }
printf("\n 最高分为:%d,最低分为:%d\n",max,min);
}
void grade(int score[],int n)      //统计各分数段的人数
{
    int i;
    int grade1=0;
    int grade2=0;
    int grade3=0;
    int grade4=0;
    int grade5=0;
    for(i=0;i<n;i++)
    {
        switch(score[i]/10)
        {
            case 10:
            case 9:grade1++;break;
            case 8:grade2++;break;
            case 7:grade3++;break;
            case 6:grade4++;break;
            default:grade5++;break;
        }
    }
    printf("\n\n 等级为优的人数:%d",grade1);
    printf("\n 等级为良的人数:%d",grade2);
    printf("\n 等级为中的人数:%d",grade3);
    printf("\n 等级为合格的人数:%d",grade4);
    printf("\n 等级为不合格的人数:%d",grade5);
}
```

为节省篇幅，任务四中已完成的主函数和主菜单函数不再给出参考代码。代码请参考任务四。

5.1 一维数组

数组，顾名思义，即一组数。在 C 语言中，数组规定了一组相同类型的数，这一组数在内存中是顺序存储的。因此，对于一个数组而言，只需要知道这个数组中第一个数据的位置和这个数组中共有多少个数据，就能一一访问这个数组的所有数据。在数组中，每一个数据称为一个元素。

对于数组的这种机制，可以举一个类似的例子来帮助读者理解：假设要通过学号来对一个班的每个人（假设点名 4 班，共有 40 人）点名，则可以这样点名：4 班的 1 号、4 班的 2 号、……、4 班的 40 号。

这个简单的例子里，其实就有类似于数组的这种用法。首先用一个班号（4 班）来表示整个班级，再用最大的人数值（40）来约束点名的人不要点超过 40 这个学号，对于班级内部的人，是用 1 号、2 号、3 号等依次递增 1 的编号来表示的，这里的每个人都是班级的一员，也就相当于是数组中的一个元素。

一个一维数组的数组名既是整个数组所有元素的统一名称，又代表该数组所有元素最开头那个元素的位置。

数组分为一维数组和多维数组。该例描述的是对一个一维数组的理解。

5.1.1 一维数组的定义

根据上面的描述，要定义一个一维数组，其实就是要指定一个名字，用它来表示一组数，同时还要指出这组数中共有多少个元素，另外还要指出这组数是什么类型的数据。

在 C 语言中使用数组必须先进行定义。

定义一维数组的一般格式为：

```
数据类型  数组名[常量表达式|符号常量]
```

在这个定义格式中，“数组名”就是一个标识符，与前面所讲的“变量”命名规则是一样的；“数据类型”是指明数组中数据是哪种类型，也就是我们熟悉的 int、float、double 和 char 等关键字；“常量表达式 | 符号常量”用来指出这个数组中最多能够存储的数据个数（即元素个数）。

说明：

（1）一维数组常常用来连续存储一组类型相同的数据，当定义一个数组 a［n］的时候，系统会自动分配一串大小为 n 的连续存储单元序列给数组 a［n］。比如 int a［n］，系统便会自动分配 n 个连续存储的整型变量存储空间给数组 a［n］。

（2）类型说明符表示数组元素的数据类型，可以为基本数据类型中的任一种。

（3）数组名的命名规则与变量相同，遵循标识符的命名规则。

（4）方括号括起来的常量表达式表示数组中元素的个数，即数组的长度。

例如：

```
int a[10]; //表示整型数组 a 有 10 个元素
float b[10],c[20]; //表示实型数组 b 有 10 个元素,实型数组 c 有 20 个元素
```

```
char ch[20]; //表示字符数组 ch 有 20 个元素
```

（5）所有数组元素共用一个名字，用下标来区别不同的元素。下标从 0 开始，按照下标顺序依次连续存放。

例如：

```
a[0],a[1],a[2],a[3],…
```

（6）常量表达式可以包含常量和符号常量，但不能是变量。C 语言中不允许对数组的大小进行动态的定义，即数组的大小不能在程序中随意改变，也不允许用浮点型数据，即不允许是 0 或负数，因为方括号里的这个值是用来表示能够存储的数的最大个数，很显然，不会出现 3.5 个数，也不会出现 0 个数或-10 个数等。

例如：如下定义是合法的。

```
int a[10];
#define N 10
int a [N],b[10],c[3+N];
```

如下定义数组的方法是错误的：

```
int n;
n=6;
int a[n];            //定义数组下标不能是变量
```

（7）允许同一个类型说明中，说明多个数组和多个变量。

例如：

```
int a,b,c,a1[10],a2[20];
```

5.1.2 一维数组的初始化

一维数组的初始化就是给数组赋值，给数组赋值的方法除了用赋值语句对数组元素逐个赋值外，还可采用初始化赋值和动态赋值的方法。

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

一维数组初始化的一般格式为：

```
类型说明符 数组名[下标]={常量表列};
```

说明：

（1）这里的“下标”表示数组元素的个数，即数组长度。

（2）“常量表列”可以是数值、字符常量或字符串常量，数组元素的初值必须依次放在一对花括号内，每个数组元素之间用逗号隔开。

（3）数组初始化时“=”后面的一对“ {} ”不能省略。

（4）“ {} ”中数值的个数不可以多于“ [] ”中指定的最大元素个数。

在应用数组的过程中常需要对数组进行初始化，方法主要有以下几种：

（1）在定义数组时对数组元素进行赋值。

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

将初值按照赋值对象的顺序排列在花括号内并以逗号隔开。

这样，数组元素 a［0］～a［9］的初值分别为 a［0］=1，a［1］=2，…，a［9］=10。

试图这样初始化是错误的做法：

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

（2）只给出一部分元素的赋值。

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

定义的数组 a 有 10 个元素，前 5 个元素 a［0］～a［4］初值分别为 a［0］=1，a［1］=2，…，a［4］=5，后 5 个元素的初值为 0。

（3）如果想对一个数组中的所有元素赋相同的初值，可以写成：

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

但不能写成：

```
int a[10]={1*10};
```

（4）对全部数组元素赋初值时，可以不指定数组长度，例如：

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

表示定义了一个数组长度为 5 的整型数组，初值分别为 a［0］=1，a［1］=2，a［2］=3，a［3］=4，a［4］=5。

若定义数组长度大于元素赋初值的个数时，不能省略数组长度的定义，而必须写成：

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

若定义数组长度小于元素赋初值的个数时，语法错误，不能执行。

5.1.3 一维数组的引用

C 语言规定，一维数组必须“先定义，后使用”，不能一次引用整个数组，只能逐个引用数组元素。

引用一维数组的一般格式为：

```
数组名[下标]
```

说明：

（1）这里的“下标”表示元素在数组中的位置，这个下标可以是整型常量或整型表达式。例如：a［1］，a［2*2］，它们分别表示数组 a 的第 2 个元素和第 5 个元素。

（2）若要对一维数组的连续多个元素引用或操作，可用一重循环实现。

例如，输出有 10 个元素的数组必须使用循环语句逐个输出各下标变量：

```
for(i=0;i<10;i++)
printf("%4d",a[i]);    //逐一输出数组 a 的每个元素
```

而不能用一个语句输出整个数组，如：

```
printf("%4d",a);        //错误
```

5.1.4 一维数组应用举例

1. 数组元素的输入与输出操作

这里将重点介绍如何利用带参数有返回值的函数来实现数组元素的输入和输出这两个功能。

【例 5.1】 输入 10 个数，然后逆序输出。

```
main()
{
```

```
    int i,t=0,a[10];
    for(i=0;i<=9;i++)
        scanf("%d",&a[i]);
    for(i=9;i>=0;i--)
        printf("%-3d",a[i]);
}
```

注意：

（1）一维数组的使用通常用一个循环来与之配合，用循环变量作为数组的下标，通过循环变量的变化（从0变到最大元素个数减1）来引用数组的每个元素。

（2）数组元素的输入和普通变量的输入一样，元素前也需要加取地址符“&”，如：scanf（"%d"，&a［i］）；

切记不能漏了地址符“&”。

【例5.2】用一维数组求10个学生的单科平均成绩。

```
main()
{
    int score[10],i,sum=0;
    float aver;
    for(i=0;i<10;i++)
    {
        scanf("%d",&score[i]);
        sum+=score[i];
    }
    aver=sum*1.0/10;
    printf("%.2f\n",aver);
}
```

注意：

因为变量sum和变量aver的类型不同，它们之间直接转换会导致精度损失，因此，在“aver=sum*1.0/10”中，sum要乘以1.0，目的是把sum从int类型转换为float类型，以确保数据的准确性。

2. 数组元素的查询操作

在较大规模程序中，数据的输入/输出可能是一次性的，更常用的操作是对这些数据进行查询。下面用两个例子来看一看数组中的最大值/最小值的查询以及如何在数组中查询一个指定的值。

【例5.3】求10个学生成绩的最高分。

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

```
    max=a[0];
    for(i=1;i<10;i++)
        if(max<a[i])max=a[i];
    printf("max=%d\n",max);
}
```

注意：

求最大值的通常做法是，先假设第一个数是最大值，并赋值给变量 max，然后将变量 max 分别跟数组后面的每一个元素进行比较，凡是大于 max 的值，就会被重新赋给 max。最后，max 就是所有值中的最大值。

上面所述是比较通用的方法，也可以通过查找最大值的下标，间接找到最大值，程序代码如下：

```
main()
{
    int a[10],i,max;
    for(i=0;i<10;i++)
        scanf("%d",&a[i]);
    max=0;     //假设最大值的所在位置为 0
    for(i=1;i<10;i++)
        if(a[max]<a[i])max=i;     //将最大值位置更新为新找到的最大值所在的位置
    printf("max=%d\n",a[max]);
}
```

【问题】

讨论与思考如何求最小值。可以参考上面的程序，请读者自行编写程序求解。

【例 5.4】 在数组中查找一个特定值。

使用例 5.2 中的若干个学生某门课程的成绩，请从这些成绩中查找到第一个不及格的成绩，输出其下标位置及该成绩。例如输入的成绩为 90、80、89、75、55、60、88 等，则应输出位置为 4，值为 55。

程序思路：

（1）先假设查找不到，即定义一个标志性变量 flag，并将其赋值为 1；

（2）查找位置指向下标为 0 的元素；

（3）比较，看成绩是否小于 60，如果是，将变量 flag 的值设置为 0，然后跳出循环，否则下标位置加 1，继续与 60 比较；

（4）判断变量 flag 的值，输出相应结果。

为了方便输入，假设该数组是 10 个学生的某门课程的成绩。

```
main()
{
    int a[10],i,flag=1;
    for(i=0;i<10;i++)
        scanf("%d",&a[i]);
```

```
    for(i=0;i<10;i++)
    {
        if(a[i]<60)
        {
            flag=0;
            break;
        }
    }
    if(flag==0)
        printf("第一个不及格的数的位置为:%d,值为:%d\n",i,a[i]);
    else
        printf("没有学生不及格\n");
}
```

【问题】

（1）如果要查找的是某个特定成绩，比如，查找第一个成绩为 75 分的，显示其下标，则程序该如何调整？

（2）如果要查找全部不及格成绩的下标及其成绩，又该怎么处理？

3. 数组元素的排序操作

有时候，需要将数组元素按照某种序列排序，这样更方便查询操作。排序有多种不同的方法，这里我们介绍三种方法，分别是普通的排序方法、选择排序法和冒泡排序法。

【例 5.5】对 10 个整数进行由小到大排序。

方法一（普通的排序方法）：

算法分析：

从第一个数开始，分别与它后面的每一个数进行比较，如果满足条件（后面的数小于第一位数），立即交换位置，一直比较到最后一个数；接下来进行第二轮比较，用第二个数分别与后面每一个数作同样的比较，依此类推，直到第九个数与其后面的数作比较。

程序代码如下：

```
main()
{
    int a[10],i,j,t;
    for(i=0;i<10;i++)
        scanf("%d",&a[i]);
    for(i=0;i<9;i++)
        for(j=i+1;j<10;j++)
        if(a[i]>a[j])
        {
            t=a[i];
            a[i]=a[j];
            a[j]=t;
        }
```

```
    for(i=0;i<10;i++)
        printf("%-3d",a[i]);
}
```

方法二（冒泡法）：

冒泡法是排序中比较经典的算法，特点是先找最大数，该算法又称“石头下沉法”。其思路是，将相邻两个数比较，将小数调到前面，大数调到后面。算法分析如下：

第一趟：10 个数从第 1 个数开始向后，将相邻两个数进行比较（共比较 9 次）。每次比较后，将小数交换到前面（将大数交换到后面）。逐次比较，经过 5 次比较与交换以后，最大的数已“沉底”。

第二趟：对余下的前面 9 个数进行第二趟比较。比较方法同第一趟，从第 1 个数向后，将相邻的两个数进行比较，将小数交换到前面（将大数交换到后面），逐次比较。经过第二趟（共 8 次比较与交换）后，得到次大的数被排列到倒数第二位。

以此类推，将最小数交换到前面后完成全部排序。

程序代码如下：

```
main()
{
    int a[10],i,j,t;
    for(i=0;i<10;i++)
        scanf("%d",&a[i]);
    for(j=0;j<9;j++)                //进行 9 次循环,实现 9 趟比较
        for(i=0;i<9-j;i++)          //在每次循环中进行 9-j 次比较
    if(a[i]>a[i+1])          //将相邻两个数比较
    {
        t=a[i];
        a[i]=a[i+1];
        a[i+1]=t;
    }
    for(i=0;i<10;i++)
        printf("%-3d",a[i]);
}
```

方法三（选择法）：

算法分析：

进行 10 次循环，第一次找到 10 个数据中的最小值，将其与第一个数据交换，然后再在剩余的 9 个数据中找出最小值将其与第二个数据交换位置。以此类推，循环到最后一个数据时，这个数必然是数组中最大的数。最后，这 10 个数被从小到大排序。

程序代码如下：

```
main()
{
    int a[10],i,j,k,t;
    for(i=0;i<10;i++)
```

```
        scanf("%d",&a[i]);
    for(i=0;i<10;i++)  //进行 10 次循环
    {
        k=i;
        for(j=i+1;j<10;j++)  //进行 10-i-1 次循环
        if(a[k]>a[j])k=j;  //记录最大值的位置
        if(k!=j)  //如果当前不是最后一个数,则将该数字与第 i 个数字交换位置
        {
            t=a[i];
            a[i]=a[k];
            a[k]=t;
        }
    }
    for(i=0;i<10;i++)
        printf("%-3d",a[i]);
}
```

4. 数组元素的移动操作

【例 5.6】将一个有 10 个元素的数组 a 的元素进行如下移动：a［0］移到 a［9］，a［1］移到 a［0］，a［2］移到 a［1］，…，a［8］移到 a［7］，a［9］移到 a［8］。

这其实就是让数组元素向前移动一个位置，最前面的元素放在最后一个位置上。关键是向前移动时要先把 a［0］的值保存好，否则移动的过程中最先丢掉的就是 a［0］了。

程序思路：

（1）先将 a［0］存储到一个变量中，如变量 b；

（2）再用循环将 a［1］到 a［9］依次向前移动一个位置；

（3）然后将变量 b 的值存储到 a［9］中，即可完成移动。

程序代码如下：

```
main()
{
    int a[10],i,k;
    for(i=0;i<10;i++)
        scanf("%d",&a[i]);
    k=a[0];              //保存好第一个数
    for(i=1;i<10;i++)
        a[i-1]=a[i];      //从第二个数起,分别向前移动一位
        a[9]=k;            //将第一个数存放在最后一位
    for(i=0;i<10;i++)
    printf("%-3d",a[i]);
}
```

5. 数组作为函数参数

（1）数组元素作为函数参数。数组元素作为函数的实参，与变量形参一样，是单向传递，即“数值传递”。只能将数组元素的值传递给被调函数的形参，不能带回变化的值。

【例 5.7】某超市的蔬菜价格会根据市场、天气情况经常变化，如某天的蘑菇、辣椒、西红柿等 10 种蔬菜价格分别为 {6，5，7，8，9，2，3，4，5，7}（单位：元），第 2 天的价格比前一天有所变化，价格分别为 {5，6，7，8，8，3，3，5，4，6}，请计算出两天蔬菜价格变化的情况。

程序分析：

可以设两个数组 a、b，大小为 10，编写一个自定义函数将它们对应的数据逐个比较，分别统计对应元素大于、等于和小于的次数。该问题可以用数组元素作为函数参数进行数据传递。

```
main()
{
    int     day1[10]={ 6,5,7,8,9,2,3,4,5,7},day2[10]={ 5,6,7,8,8,3,3,5,4,6};
    int i,f,n=0,m=0,k=0;
    for(i=0;i<10;i++)
    {
        f=compare(day1[i],day2[i]);
        if(f==1)n++;
        else if(f==0)m++;
            else k++;
    }
    printf("第一天高于第二天价格:%d 项\n",n);
    printf("第一天等于第二天价格:%d 项\n",m);
    printf("第一天小于第二天价格:%d 项\n",k);
}
```

（2）数组名作为函数参数。数组名表示数组的首地址，数组名本身就是指针。因此，用数组名作为函数参数，是把数组的首地址作为实参传递给被调用函数形参，是指针传递参数。

【例 5.8】用数组名作为函数参数调用，求 10 个学生成绩的平均分及所有高于平均分的成绩。

```
float average(float a[10])
{
    int i;
    float sum=0,avg;
    for(i=0;i<10;i++)
        sum=sum+a[i];
    avg=sum/10;
    return avg;
}
void max(float avg,float x[10])
{
    int i;
    for(i=0;i<10;i++)
```

```
    {
        if(avg<x[i])
        printf("%f\n",x[i]);
    }
}
main()
{
    float a[10],avg;
    int i;
    for(i=0;i<=9;i++)
        scanf("%f",&a[i]);
    avg=average(a);
    printf("平均分是:%f\n",avg);
    max(avg,a);
}
```

5.2 二维数组

如果要处理30个学生5门课程的成绩，首先要解决的问题是如何在计算机中表示这些成绩数据。我们可以使用5个一维数组，每个数组包含30个元素，但是这样一来每个学生的成绩处于分隔状态，很难理清同一个学生5门课程的位置，程序处理起来很复杂。

更好的方法是采用数组的数组，即主数组包含30个元素，每个元素代表一个学生成绩。代表一个学生成绩的数组是包含5个元素的数组。这种数组的数组称为二维数组。

5.2.1 二维数组的概念

二维数组可以用来表示生活中像矩阵这样的二维数据，如学生的多科成绩统计表、学生基本信息登记表等，但是其数据的存储本质上还是一组连续存储单元。实际上，二维数组在内存中的逻辑存储与一维数组一样，呈现线性排列，即第一行的数据之后，紧跟着第二行的数据。这一点我们要注意，二维数组的逻辑存储结构和将二维数组理解为矩阵是不同的。

5.2.2 二维数组的定义

二维数组实际上是一维数组的一维数组，即它的每一行都是一个一维数组。

定义二维数组的一般格式为：

类型说明符 数组名[常量表达式1][常量表达式2]

说明：

（1）数组名后“[常量表达式]”个数表示数组的维数，C语言允许定义和使用多维数组。

（2）对于二维数组，第一个“[常量表达式]”表示二维数组的“行”，第二个“[常量表达式]”表示二维数组的“列”。如“int a[3][4];”就表示定义了一个数组名为a的3×4（3行4列）的二维数组。

（3）二维数组中元素的表示方法：以“int a[3][4];”为例，可以将二维数组看成

是一种特殊的一维数组，它里面的元素又是一些一维数组。如可以将二维数组 a 看作是只有 3 个元素的一维数组，即 a［0］、a［1］、a［2］，这 3 个元素又是各自拥有 4 个元素的一维数组。结构为：

$$a[3][4]\begin{cases} a[0]\rightarrow a[0][0] & a[0][1] & a[0][2] & a[0][3] \\ a[1]\rightarrow a[1][0] & a[1][1] & a[1][2] & a[1][3] \\ a[2]\rightarrow a[2][0] & a[2][1] & a[2][2] & a[2][3] \end{cases}$$

（4）C 语言中二维数组的元素在内存中的存储顺序是按行存放的，即先存储第一行，接着存储第二行，依此类推。如 a［3］［4］中 12 个元素的存储顺序为：

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

（5）多维数组的定义规则和二维数组类似，如：

```
float a[3][3][4];
```

表示定义了一个三维数组，元素个数为 3×3×4＝36 个，其中各元素在内存中存放的顺序为：

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

5.2.3 二维数组的初始化

二维数组初始化也是在类型说明时给各下标变量赋以初值。二维数组可按行分段赋值，也可按行连续赋值。

二维数组初始化的方法主要有以下几种：

（1）分行给二维数组赋初值。如：

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

把第 1 个花括号内的数据赋给第一行的元素 a［0］［0］、a［0］［1］、a［0］［2］、a［0］［3］，第 2 个花括号内的数据赋给第二行的元素 a［1］［0］、a［1］［1］、a［1］［2］、a［1］［3］……即按行赋初值。

（2）可以将所有数据写在一个花括号内，系统将按数组排列顺序对各个元素赋值。如：

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

这种赋值方法与第一种类似，但不直观，数据太多时容易遗漏，也不易检查。

（3）可以对部分元素赋值。

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

其作用是对每行第 1 列数据进行赋值，其余元素自动为 0。赋值后各元素的值为：

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 2 & 0 & 0 & 0 \\ 3 & 0 & 0 & 0 \end{bmatrix}$$

也可以对各行中的某一个或几个元素进行赋值：

```
int a [3] [4] = { {1}, {0, 2}, {0, 0, 0, 9} };
```

这样初始化后，各元素的值为：

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 0 & 9 \end{bmatrix}$$

这样的初始化方法适用于非 0 元素较少时的赋值，不必将所有的 0 写出来，只需输入少量的数据即可。

也可以只对某几行元素赋初值：

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

初始化后，各元素的值为：

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

没有对第 3 行进行赋值，取值自动为 0，也可以对第 2 行不赋值：

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

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

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

等价于：

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

系统会根据数据的总个数来分配存储空间：一共 12 个数据，每行 4 个，显然能确定一共 3 行。

在定义时也可以只对部分元素赋初值而省略第一维的长度，但应分行赋值。如：

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

这种写法也能告诉编译系统，数组共有 3 行，赋值后各元素的值为：

$$\begin{bmatrix} 1 & 2 & 3 & 0 \\ 0 & 0 & 0 & 0 \\ 4 & 5 & 0 & 0 \end{bmatrix}$$

5.2.4 二维数组的引用

二维数组的元素也称为双下标变量，引用的时候必须有双下标。

引用二维数组的一般格式为：

数组名[下标][下标]

说明：

（1）下标应该为整型常量或整型表达式。

如 a［2］［3］，表示第 3 行第 4 列的元素。下标也可以是整型表达式，如 a［1＊2］［3-1］。不能写成 a［2，3］或 a［1＊2，3-1］这样的错误形式。

（2）数组元素可以出现在表达式中，也可以被赋值，如：

b［1］［1］=a［1］［1］＊2

（3）在使用数组元素时，应注意下标值在已定义的数组大小范围之内，取值从 0 开始，数组中最后一个元素的下标应为定义数组的［常量表达式］-1。例如：

int a［3］［4］;

那么，数组中最后一个元素为 a［2］［3］，a［3］［＊］或 a［＊］［4］是不存在的。因此读者应注意区分定义中的 a［3］［4］和引用中的 a［3］［4］，前者是定义数组的维数和大小（表示定义了一个 3 行 4 列的二维数组），后者是一个元素（表示第 4 行第 5 列的数组元素）。注意：a［3］［4］不是所定义的数组 a［3］［4］中的元素。

5.2.5 二维数组应用举例

【例 5.9】使用二维数组求如下矩阵的转置矩阵：

$$\boldsymbol{a}=\begin{bmatrix}1 & 2 & 3\\ 4 & 5 & 6\end{bmatrix}\Rightarrow\boldsymbol{b}=\begin{bmatrix}1 & 4\\ 2 & 5\\ 3 & 6\end{bmatrix}$$

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

注意：

遍历二维数组时，常用两个 for 循环，第一个 for 循环用于遍历行，第二个 for 循环用于遍历列。

【例 5.10】有一个 3×4 阶的矩阵，要求编程求出其中值最大的那个元素的值，以及其所在的行号和列号。

```
int max_val(int a[3][4])
{
    int i,j,max,row,column;
    max=a[0][0];
    for(i=0;i<3;i++)
        for(j=0;j<4;j++)
            if(max<a[i][j])
    {
        max=a[i][j];
        row=i;
        column=j;
    }
    printf("%d,%d,%d\n",max,row,column);
}
main()
{
    int a[3][4]={{2,4,6,8},{9,5,3,1},{12,1,67,10}},max;
    maxval(a);
}
```

注意：

在遍历二维数组过程中，如果数组元素 a［i］［j］大于变量 max 的值时，除了将 a［i］［j］的值赋给 max，还必须将当前数组元素的行号 i 和列号 j 分别赋给变量 row 和 column，这样才能确保最终变量 row 和 column 是最大值的行号和列号。

5.3 字符数组

字符数组就是类型为 char 的数组，用于存储一串连续的字符。字符数组中每个数组元素存放的值都是单个字符（1 个字节），这些单个字符连续存储就构成了字符数组，字符数组既可以是一维的，也可以是多维的。

5.3.1 字符数组的定义

用来存放字符数据的数组是字符数组。字符数组中的一个元素存放一个字符。字符数组的定义方法与前面介绍的数组定义方法类似。如：

```
char c[12];
```

定义了一个字符数组，它有 c［0］、c［1］、c［2］、c［3］、c［4］、c［5］、c［6］、

c［7］、c［8］、c［9］、c［10］、c［11］共 12 个元素，可以给它们赋值，例如：

```
c[0]='H';c[1]='e';c[2]='l';c[3]='l';c[4]='o';c[5]=' ';c[6]='w';c[7]='o';c[8]='r';c[9]='l';c[10]='d';c[11]='!';
```

它们的存储状态为：

H	e	l	l	o		w	o	r	l	d	!

也可以使用整型数据类型来定义字符数组：

```
int c[5];
```

但这样定义将本来一个字节的字符数据采用四个字节来存放，浪费了存储空间。

也可以定义二维字符数组：

```
char c[2][5];
```

5.3.2 字符数组的初始化

字符数组的初始化方法与数组初始化方法规则类似，既可以给每个元素单独赋值，也可以定义时就直接对每个元素赋值，如：

```
char c[12]={'H','e','l','l','o',' ','w','o','r','l','d','!'};
```

如果初值个数大于数组定义的范围，则编译器会提示语法错误。如果初值个数小于数组范围，则将初值按顺序赋给前面的元素，剩下的元素自动定义为空字符（‘\0’）。如：

```
char c[8]={'H','e','l','l','o',' ','c'};
```

数组赋值后状态为：

c[0]	c[1]	c[2]	c[3]	c[4]	c[5]	c[6]	c[7]
H	e	l	l	o		c	\0

二维字符数组与二维数组的初始化方法类似：

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

这个 5 行 5 列字符数组组成了一个钻石平面图形。

5.3.3 字符数组的引用

用字符数组的下标指定要引用的数组元素。

一维字符数组的引用格式：

```
数组名[下标]
```

二维字符数组的引用格式：

```
数组名[行下标][列下标]
```

【例 5.11】 打印输出一个字符串。

```
main()
{
    char c[10]={'I',' ','a','m',' ','a',' ','b','o','y'};
    int i;
    for(i=0;i<10;i++)
```

```
        printf("%c",c[i]);
    printf('\n');
}
```

5.3.4 字符串

用双引号引起来的一串字符称为字符串，如"xyz"。在C语言中，字符串是存储在字符型数组中的。存放字符串时，系统会自动在有效字符（如xyz）后面加上'\0'字符。'\0'是ASCII为0的转义字符，它是字符串的结束标志。

在字符串中，有效字符的个数称为“字符串的长度”，但实际上，字符串在内存中所占的字节数会比字符串的长度多1（因为'\0'要占一个字节）。例如字符串"xyz"的长度为3，但它在内存中的字节数是4。再例如：char c[ ] = {“Hello”}；或者省略花括号：char c[ ] = “Hello”；

在系统内存中存放情况为：

H	e	l	l	o	\0

注意：字符串数组c的数组长度为6（实际长度为5），这与如下的字符数组是等价的：

```
char c[ ]={'H','e','l','l','o','\0'};  //注意是单引号,数组长度为6
```

而与下面的数组是不同的：

```
char c[ ]={'H','e','l','l','o' };  //数组长度为5
```

如果有：

```
char c[10]=“Hello”;
```

数组c的前5个字符是'H''e''l''l''o'，第6个字符是'\0'，后4个字符为空字符。其在内存中的存储情况为：

H	e	l	l	o	\0	\0	\0	\0	\0

要说明的是，字符串数组并不要求它的最后一个字符为'\0'，甚至可以不包括'\0'，比如以下的字符串数组定义和初始化是完全合法的：

```
char c[5]={ 'H','e','l','l','o'};
```

是否使用'\0'根据需要而定。但由于系统对字符串常量总是自动加一个'\0'，因此为了使处理方法一致，便于测定字符串的实际长度，以及在程序中作相应的处理，在字符数组中也常常人为地加上一个'\0'。如：

```
char c[ ]={'H','e','l','l','o','\0'};     //也可以不加\0
```

5.3.5 字符数组的输入/输出

字符数组的输入/输出主要有两种方法：

（1）用“%c”格式符逐个输入/输出。

（2）用“%s”格式符将整个字符串（String）一次输入/输出。

如：

```
char c[6];
scanf("%c",&c[0]);          //从键盘输入一个字符存放在 c[0]中
printf("%c",c[0]);          //输出字符 c[0]
scanf("%s",c);              //输入字符串存放在字符串数组 c 中,此处不需要用 &c,因为数组名
                              本身就是地址
printf("%s",c);             //输出字符串 c
```

注意：

（1）输出时，遇'\0'结束，且输出字符中不包含'\0'。

（2）用“%s”格式输出字符串时，printf（）函数的输出项是字符数组名，而不是元素名。

例：

```
char c[6] = "China";
```

输出时以下两种写法都是对的：

```
printf("%s",c);
printf("%c",c[0]);
```

以下写法是不对的：

```
printf("%s",c[0]);
```

（3）用“%s”格式输出时，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:China */
```

（4）用“%s”格式输出时，若数组中包含一个以上'\0'，遇第一个'\0'时结束。

（5）输入时，遇回车键结束，但获得的字符中不包含回车键本身，而是在字符串末尾添'\0'。因此，定义的字符数组必须有足够的长度，以容纳所输入的字符（如，输入 5 个字符，定义的字符数组至少应有 6 个元素）。

（6）一个 scanf 函数输入多个字符串，输入时以“空格”键作为字符串间的分隔。

例：

```
char str1[5],str2[5],str3[5];
scanf("%s%s%s",str1,str2,str3);
```

输入数据：How are you?↙

str1、str2、str3 被赋值的数据为：

H	o	w	\0	
a	r	e	\0	
y	o	u	?	\0

若改成：

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

输入：How are you?

结果：仅“How”被输入数组 str，str 的赋值情况为：

如要想 str 获得全部输入（包含空格及其以后的字符），程序可设计为：

```
charstr[13];
int i;
for(i=0;i<13;i++)
str[i] = getchar();
```

或者：

```
char str[13];
gets(str);
```

（7）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);
```

以下写法就是不对的：

```
scanf("%s",&str);
```

如例 5.11，也可以用"%s"输出整个字符串，代码如下：

```
main()
{
    char c[11]={'I',' ','a','m',' ','a',' ','b','o','y'};
    int i;
    c[10]='\0';
    printf("%s",c);
    printf('\n');
}
```

注意：

字符串要以'\0'作为结束标志，所以，字符数组 c 的数组长度更改为 11，c［10］用来存储'\0'。

5.3.6 字符串处理函数

为了方便编程者对字符串进行处理，在 C 编译器的函数库中包含了一些字符串函数，熟练地利用这些函数，对于程序的字符串处理是很有好处的。使用输入/输出字符和字符

串函数时，应在函数前加上头文件“stdio. h”；使用其他字符串操作函数时，还应加上头文件“string. h”。下面介绍几种常用的函数。

1. puts（字符数组）

作用是将一个字符串（以'\0'结束）输出到终端。如果已定义名为 str1 的字符串数组，且初始化为“hello world!”，则应用

```
puts(str1);
```

其结果是在终端上输出 hello world!。由于可以用 printf 函数输出字符串，因此 puts 函数用的并不多。用 puts 函数可以输出带转义字符的字符串。例如：

```
char str[ ] = {"hello\nworld!  \n"};
puts(str);
```

输出：

```
hello
world!
```

在输出时将字符串结束标志'\0'转换成'\n'，即输出完字符串后换行。

2. gets（字符数组）

作用是从终端输入一个字符串到字符串数组，并返回一个表示字符数组起始地址的函数值。如执行下面的函数：

```
gets(str);
```

从键盘输入“computer”，回车。

其结果是将'c''o''m''p''u''t''e''r''\0'这 9 个字符送给字符串数组 str，函数返回值是字符串数组 str 的起始地址。当然，一般利用 gets 函数的目的是向字符串数组 str 输入一个字符串，而并不关心其函数返回值。

注意，puts 和 gets 函数只能输入和输出一个字符串，不能写成：

```
puts(str1,str2)或 gets(str1,str2)
```

【例 5.12】从一个字符串中截取出指定长度的子字符串。

```
main( )
{
    char a[40],b[40];
    int i,m;
    gets(a);
    scanf("%d",&m);
    for(i=0;i<m;i++)
        b[i]=a[i];
    b[i]='\0';          // b[m]='\0';
    puts(b);
}
```

注意：

把字符数组a的前m个字符复制到字符数组b中之后，必须设置b[i]='\0'，或者b[m]='\0'，因为'\0'是字符串的结束标志。

在C语言中，输入/输出函数总结：

（1）getchar（）和putchar（）主要用于输入和输出单个字符，并且每次只能输入和输出一个字符。

（2）gets（）和puts（）主要用于输入和输出字符串，并且每次只能输入和输出一个字符串。

（3）scanf（）和printf（）只能设置好格式控制，可以输入和输出任意类型的数据。如%d用于控制输入和输出带符号的十进制整型数据，%f用于控制输入和输出带小数点的数据，%c用于控制输入和输出字符型数据。%s用于控制输入和输出字符串……

【例5.13】输入一行字符，统计其中有多少个单词，单词之间用空格分隔开。

```
main()
{
    char a[40];
    int i,n=0,flag=0;
    gets(a);
    for(i=0;a[i]!='\0';i++)
        if(a[i]==' ')flag=0;
        else if(flag==0)
    {
        flag=1;
        n++;
    }
    printf("%d\n",n);
}
```

3. strcat（字符串数组1，字符串数组2）

strcat是string catenate（字符串连接）的缩写，作用是连接两个字符串数组中的字符串，把字符串2串接到字符串1的后面，串接成的新字符串存放在字符串数组1中。函数调用后得到一个表示字符串数组1地址的返回值。例如：

```
char str1[20]={"I am "};
char str2[ ]={"a Chinese"};
printf("%s",strcat(str1,str2));
```

连接前后的数组内容为：

str1	I		a	m		\0													
str2	a		C	h	i	n	e	s	e	\0									
str1	I		a	m		a		C	h	i	n	e	s	e	\0				

注意：

（1）字符串数组 1 必须有足够的长度以装下串接后的新字符串。

（2）连接前两个字符串的后面都有一个'\0'，连接后将第一个字符串后面的'\0'去掉，只在最后保留一个'\0'。

4. strcpy（字符数组 1，字符数组 2）

strcpy 是 string copy（字符串复制）的缩写。作用是将字符串 2 复制到字符串数组 1 中去。例如：

```
char str1[10],str2[ ]={"hello!"};
```

执行：

```
strcpy(str1,str2);
```

str1 的状态为：

h	e	l	l	o	!	\0

注意：

（1）字符串数组 1 必须有足够的长度以容纳复制进来的字符串。字符串数组 1 的长度应不小于字符串数组 2 的长度。

（2）“字符串数组 1”必须写成数组名形式（如 str1），“字符串数组 2”可以是字符串数组名，也可以是一个字符串常量，如：

strcpy（str1，“hello!”）；

（3）复制时连同字符串后面的'\0'一起复制到字符串数组 1 中。

（4）不能用赋值语句将一个字符串常量或字符串数组赋给一个字符串数组，如下面两种写法都是不合法的：

```
str1={"hello!"};
str1=str2;
```

而只能用 strcpy 函数进行字符串的整体赋值。用赋值符号只能将一个字符赋给一个字符型变量或字符数组元素。

（5）可以用 strcpy 函数将字符串 2 前面若干个字符复制到字符数组 1 中去。如：

strcpy（str1，str2，2）；

作用是将 str2 中前面 2 个字符复制到 str1 中去，取代 str1 中前面 2 个字符。

5. strcmp（字符串 1，字符串 2）

strcmp 是 string compare（字符串比较）的缩写。功能是对字符串进行比较，并返回一个结果。

例如：

```
strcmp(str1,str2);
strcmp(str1,"beijing");
strcmp("shanghai","wuhan");
```

函数对字符串的比较规则为：对两个字符串自左至右逐个字符的 ASCII 码进行比较，直到出现不同的字符或遇到'\0'为止。如果字符串全部相同则认为相等。若出现不同的

字符，则以第一个不同的字符的比较结果为准。如：

'A'<'B','C'<'c','3'>'!',"girl">"boy","Women"<"men"

比较的结果由函数返回值带回。具体规则如下：

（1）如果两字符串相等，函数返回值为0；

（2）如果字符串1>字符串2，函数返回值为正数；

（3）如果字符串1<字符串2，函数返回值为负数；

注意：

判断两个字符串，不能采用以下的方法：

```
if（str1==str2）
printf（string1 is equal to string2）;
else
…
```

而应该采用strcmp函数，根据返回值来判断：

```
if（strcmp（str1，str2））printf（"…"）;
```

【例5.14】有3个字符串，要求找出其中最大值。

```
main()
{
    char string[20],str[3][20];
    int i;
    for(i=0;i<3;i++)
        gets(str[i]);
    if(strcmp(str[0],str[1])>0)
        strcpy(string,str[0]);
    else
        strcpy(string,str[1]);
    if(strcmp(str[2],string)>0)
        strcpy(string,str[2]);
    printf("%s",string);
}
```

也可以直接用一维数组解决上述例题。

```
main()
{
    char string[20],str1[20],str2[20],str3[20];
    int i;
    gets(str1);
    gets(str2);
    gets(str3);
    if(strcmp(str1,str2)>0)
        strcpy(string,str1);
```

```
    else
        strcpy(string,str2);
    if(strcmp(str3,string)>0)
        strcpy(string,str3);
    printf("%s",string);
}
```

6. strlen（字符数组）

strlen 是 string length（字符串长度）的缩写。作用是测试字符串的长度，函数返回值为字符串的实际长度（有效字符个数），不包括‘\0’在内。如：

```
char str1[15]={"hello world"};
```

运行：

```
printf("%s",strlen(str1));
```

输出结果为 11。

【例 5.15】某单位的工作证号的最后一位是用来表示性别的，M 表示男性，F 表示女性。输入 5 个人的工作证号码，请统计出其中的男女人数。

```
main()
{
    int a=0,b=0,i,n;
    char c,s[5][10];
    printf("请输入 5 个人的工作证号码:\n");
    for(i=0;i<5;i++)
    {
        scanf("%s",s[i]);
    n=strlen(s[i]);
        c=s[i][n-1];
    if(c=='m'||c=='M')
        a++;
    else
        b++;
    }
    printf("男生数为:%d\n 女生数为:%d\n",a,b);
}
```

【例 5.16】输入一个字符串，判断它是否是一个回文串。所谓回文串是指这个字符串从左到右及从右到左的字母排列是一样的，例如“aba”“abcba”等。

程序分析：

判断一个字符串 s 左右顺序是否一样，可以设计一个从左到右的下标 i，i=0、1、…、strlen（s）-1，同时设计一个从右到左的下标 j，j=strlen（s）-1、strlen（s）-2、…、1、0，每次比较 s［i］==s［j］，则++i，--j，再比较下一对，如 s［i］!=s［j］，则 s 肯定不是回文串，如对所有的 i<j，都有 s［i］==s［j］，则 s 是回文串。

程序代码如下：

```
main( )
{
    char s[100];
    int i=0,j=0;
    printf("输入字符串:\n");
    gets(s);
    i=0;
    j=strlen(s)-1;
    while(i<j)
    {
        if(s[i]==s[j])
        {
            i++;
            j--;
        }
        else
        {
            puts("不是回文串");
            break;
        }
    }
    if(i>=j)
        puts("是回文串");
}
```

7. strlwr（字符串）

strlwr 是 string lowercase（字符串小写）的缩写，作用是将字符串中大写字母换成小写字母。

8. strupr（字符串）

strlpr 是 string uppercase（字符串大写）的缩写，作用是将字符串中小写字母换成大写字母。

【例 5.17】输入一个字符串，将其中的大写字母均换成小写字母。

```
main( )
{
    char a[40];
    gets(a);
    strlwr(a);
    puts(a);
}
```

任务六　学生成绩管理系统中成绩的统计（2）

学习目标

- 理解和掌握指针的含义；
- 掌握指针的定义、初始化和使用方法；
- 了解指针和数组的关系。

一、任务描述

该任务利用指向数组的指针来实现任务五中的自定义函数的功能，即用另一种方法来实现函数的功能。

二、知识要点

使用指针来实现函数间的通信（即作函数参数），也可以通过指针来灵活地操作数组和字符串。

三、任务分析

在任务五中用一个一维数组 score［］来存放学生成绩，并作为函数的形参来传递，在函数体中对数组元素的访问采用的是下标法。本任务中依旧需要使用一个 st_score［］来存放学生成绩，然后定义一个指针指向该数组的首地址。将函数中的形参改为指针，则函数体中对数组元素的访问，也相应修改为指针方式。

四、源代码参考

```
/*--------------------项目的整体框架实现-------------*/
/*==========================预处理命令==========================*/
#include<stdio.h>
#include<stdlib.h>
#include<conio.h>
#include<string.h>
#define MAXSTU 30 //学生人数最大为 30
/*==========================函数原型声明==========================*/
void login();                              //密码验证函数声明
void menu();                               //主菜单函数声明
int  input(int *score);                    //录入学生成绩函数声明
void output(int *score,int  n);            //显示学生成绩函数声明
void SumAvg(int *score,int n);             //统计课程总分和平均分函数声明
void MaxMin(int *score,int n);             //统计课程最高分和最低分函数声明
void grade(int *score,int n);              //统计课程各分数段人数函数声明
```

```
/*==========主函数==========*/
void main()                                    //主函数
{
    int stscore[MAXSTU];                       //定义一维数组,存放学生某门课程的成绩
    int *score=stscore;                        //定义一个指向成绩数组的指针
    int count=0;                               //存放学生实际人数
    int choose;                                //定义整型变量,存放主菜单选择序号
    login();                                   //调用密码验证函数
    while(1)
    {
        menu();                                //调用显示主菜单函数
        printf("\t\t 请选择主菜单序号(0-5)");
        scanf("%d",&choose);
        switch(choose)
        {
            case 1:count=input(score);         //调用录入学生成绩函数
                break;
            case 2:output(score,count);        //调用显示学生成绩函数
                break;
            case 3:SumAvg(score,count);        //调用统计总分和平均分函数
                break;
            case 4:MaxMin(score,count);        //调用统计最高分和最低分函数
                break;
            case 5:grade(score,count);         //调用统计各分数段人数函数
                break;
            case 0:return;                     //退出当前函数或程序
                default:printf("\n\n\n 输入无效请重新选择\n");
        }
            printf("\n\n\n 按任意键返回主菜单");
            getch();
    }
}
/*==========函数定义部分==========*/
void login()                                   //登录函数
{
    char pwd[10]="abc123";
    char ch[10];
    int re;
    printf("请输入密码:\n");
    gets(ch);
    re=strcmp(ch,pwd);
```

```
    if(re==0)
        puts("密码正确,登录成功");
    else
    {
        puts("密码不正确,请重新输入:");
        login();
    }
}
void menu()                                    //主菜单函数
{
    system("cls");
    printf("\n\n");
    printf("\t\t ************************************************************************\n");
    printf("\t\t                        学生成绩管理系统                                \n");
    printf("\t\t ************************************************************************\n");
    printf("\t\t                        1—录入学生成绩                                  \n");
    printf("\t\t                        2—显示学生成绩                                  \n");
    printf("\t\t                        3—统计总分和平均分                              \n");
    printf("\t\t                        4—统计最高分和最低分                            \n");
    printf("\t\t                        5—统计各分数段人数                              \n");
    printf("\t\t                        0—退出                                          \n");
    printf("\t\t ************************************************************************\n");
}
int input(int *score)                          //输入学生成绩函数
{
    int i;
    printf("\n 输入学生成绩(输入-1 退出)\n");
    for(i=0;i<MAXSTU;i++)
    {
        printf("\t 第%d 个学生的成绩为:",i+1);
        scanf("%d",score+i);
        if(*(score+i)==-1)
            break;
    }
return i;                                      //返回学生的实际人数
}
void output(int *score,int n)                  //显示学生成绩函数
{
    int i;
    printf("\n\n 学生成绩:");
    printf("\n 学号\t\t 成绩");
```

```
    for(i=0;i<n;i++)
    {
        printf("\n%d\t\t%d",1001+i,*(score+i));
    }
}
void SumAvg(int *score,int n)                    //统计总分和平均分
{
    int i,sum=0;
    float ave=0;
    for(i=0;i<n;i++)
    {
        sum+=*(score+i);
    }
    ave=(float)sum/n;
    printf("\n 总分为%d,平均分为%.2f\n",sum,ave);
}
void MaxMin(int *score,int n)                    //统计最高分和最低分
{
    int i,max=0,min=0;
    max=*score;
    min=*score;
    for(i=0;i<n;i++)
    {
        if(*(score+i)>max)
            max=*(score+i);
        if(*(score+i)<min)
            min=*(score+i);
    }
    printf("\n 最高分为:%d,最低分为:%d\n",max,min);
}
void grade(int *score,int n)                     //统计各分数段的人数
{
    int i;
    int grade1=0;
    int grade2=0;
    int grade3=0;
    int grade4=0;
    int grade5=0;
    for(i=0;i<n;i++)
    {
        switch(*(score+i)/10)
```

```
            {
                case 10:
                case 9:grade1++;break;
                case 8:grade2++;break;
                case 7:grade3++;break;
                case 6:grade4++;break;
                default:grade5++;break;
            }
        }
        printf("\n\n 等级为优的人数:%d",grade1);
        printf("\n 等级为良的人数:%d",grade2);
        printf("\n 等级为中的人数:%d",grade3);
        printf("\n 等级为合格的人数:%d",grade4);
        printf("\n 等级为不合格的人数:%d",grade5);
    }
```

6.1 指针和指针变量

指针是 C 语言中广泛使用的一种数据类型，指针的使用极大地丰富了 C 语言的功能，运用指针编程是 C 语言最主要的风格之一。利用指针变量可以表示各种数据结构，能很方便地使用数组和字符串，并能像低级语言一样处理内存地址，从而编写出精练而高效的程序。

学习指针是 C 语言学习中最重要的环节之一。只有深入学习和掌握指针，才能领会到 C 语言的精华所在，能否正确理解和使用指针是衡量是否掌握 C 语言的一个重要标志。但指针学习也是 C 语言中最为困难的一部分，在学习中除了要正确理解基本概念，还必须要多编程，多上机调试。只要做到这些，指针也是不难掌握的。

6.1.1 指针的基本概念

1. 变量的指针与变量的值

计算机中内存是以字节为单位的一片连续的存储空间，每一个字节都有一个编号，这个编号就是内存地址。就像教学大楼的房间编号一样，根据房间号可以很容易地找到教室及其中的某位同学，同样，根据内存地址也可以准确地找到相应的内存单元及其中的变量值。

C 语言程序中的任意一个变量，在定义为某一种类型后，编译系统会根据该变量的类型分配存储单元而使其得到地址。例如，整型变量与实型变量都是分配 4 个字节的存储空间，字符型变量分配 1 个字节的存储空间。变量在内存的起始地址，又称为变量的指针，变量的值就被放入所分配的地址单元中，从图 6-1-1 可以看出变量的值与变量的地址之间的对应关系（就如同大楼中某个房间住的人和其房间编号的关系）。

指针（地址）	内存	字节数	变量名
2000H	10	4	a
2004H	20	4	b
2005H	X	1	c
2006H	Y	1	d
200AH	5.6	4	e
200EH	-2.8	4	f

图 6-1-1　变量与地址

【例 6.1】按要求输出变量的值及变量的地址值。

```
#include "stdio.h"
main()
{
    int a=10,b=20;
    char c='X',d='Y';
    float e=5.6,f=-2.8;
    printf("%d,%d\n",a,b);      /* 输出整型变量 a,b 的值 */
    printf("%c,%c\n",c,d);      /* 输出字符变量 c,d 的值 */
    printf("%f,%f\n",e,f);      /* 输出实型变量 e,f 的值 */
    printf("%u,%u,%u,%u,%u,%u\n",&a,&b,&c,&d,&e,&f); /* 输出所有变量的地址值 */
}
```

运行结果如图 6-1-2 所示。

```
10,20
X,Y
5.600000,-2.800000
2293532,2293528,2293527,2293526,2293520,2293516

--------------------------------
Process exited with return value 0
Press any key to continue . . .
```

图 6-1-2　例 6.1 运行结果

指针又称地址，指针变量即存放地址的变量。变量的三要素是变量名、变量值与变量地址。而变量值与变量地址是两个不同的概念，变量在内存所占的存储单元中存放的数据称为变量的值；变量在内存所占的存储单元的首地址称为变量的地址。变量的地址只可以存储在指针变量中，通过指针变量来引用地址中的数据。

指针变量与其他变量一样，也要先定义后使用。

指针变量定义的一般形式：

```
类型说明符 *变量名;
```

指针变量的定义中包括三个内容：

（1）指针类型说明，即定义变量为一个指针变量；

（2）指针变量名；

（3）变量值（指针）所指向的变量的数据类型。

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

```
int *p1;
```

p1 是一个指针变量，它的值是某个整型变量的地址，或者说 p1 指向一个整型变量。至于 p1 究竟指向哪一个整型变量，应由向 p1 赋予的地址来决定。

再如：

```
float *p2;          /*p2 是指向浮点变量的指针变量*/
static int *p3;     /*p3 是指向静态整型变量的指针变量*/
char *p4;           /*p4 是指向字符变量的指针变量*/
```

必须注意的是，一个指针变量只能指向相同类型的变量，如 p2 只能指向 float 类型的变量，不能时而指向一个 float 类型的变量，时而又指向一个 int 或 char 类型的变量。

指针变量和普通变量一样，在使用前要先定义说明，然后必须进行赋值。未经赋值的指针变量不能使用，否则将造成系统混乱，甚至死机，因为在定义指针变量而没有给它赋值之前，指针变量的值是不定的。指针变量只能获得地址值，绝不能赋予任何其他数据，否则将引起错误。

C 语言中提供的地址运算符“&”可以用来表示变量的地址。

其一般形式为：

```
&变量名;
```

如：&a 表示变量 a 的地址，&b 表示变量 b 的地址。

设已有整型变量 a 和指向整型变量的指针变量 p，若要把整型变量 a 的地址赋予 p，即令指针变量 p 指向整型变量 a，则可以有以下两种方式实现：

（1）赋值语句的方法。

```
int a;
int *p
p=&a;
```

（2）指针变量初始化的方法。

```
int a;
int *p=&a;
```

不允许把一个数赋予指针变量，因此下面的赋值是错误的：

```
int *p;
p=1000;
```

被赋值的指针变量前不能再加“*”说明符，如写为 *p=&a 也是错误的。

6.1.2 指针变量的引用和运算

指针变量是存放目标变量地址的变量，因此可以通过指针变量间接存取目标变量。指针变量可以进行某些运算，但其运算的种类是有限的。它只能进行赋值运算和部分算术运算及关系运算。

1. 指针运算符

（1）取地址运算符 &。

取地址运算符 & 是单目运算符，其结合性为自右至左，其作用是获取变量的地址。

（2）取内容运算符 * 。

取内容运算符 * 是单目运算符，其结合性为自右至左，用来表示指针变量所指的变量。在 * 运算符之后跟的变量必须是指针变量。

需要注意的是，指针运算符 * 和指针变量定义中的指针说明符 * 不是一回事。在指针变量说明中，“ * ”是类型说明符，表示其后的变量是指针类型。而表达式中出现的“ * ”则是一个运算符，用以表示指针变量所指的变量。

【例 6.1】通过指针访问简单变量。

```
#include <stdio.h>
int main()
{
    int a=12, *p=&a;
    printf("a=%d\n", *p);
    return 0;
}
```

语句“int a=12, *p=&a;”中的 *p 表示定义了一个指向整型变量的指针变量 p，且指针变量 p 取得了整型变量 a 的地址。语句“printf("%d", *p);”中的 *p 表示 p 所指向的变量中的内容，指针变量 p 中存放了 a 的地址，故本语句表示输出指针变量 p 所指向的变量 a 的值，结果应为 12，程序运行结果如图 6-1-3 所示。

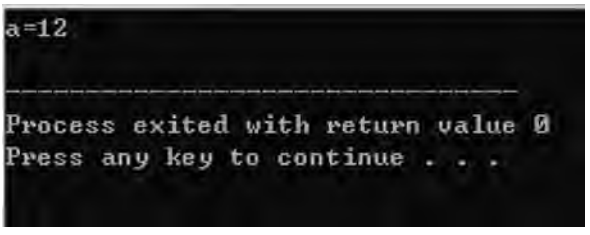


图 6-1-3　例 6.1 程序运行结果

若已有“int a=12, *p=&a;”定义，那么“&*p”和“*&a”的含义是什么呢？

“&”和“ * ”运算符的优先级别相同，其结合性为自右至左，故“&*p”中，应先执行“ *p”的运算，它的结果就是变量 a，再执行 & 运算，即执行“&a”，所以“&*p”就是“&a”，结果为 a 的地址。

同理，在执行“ *&a”时，应先执行“&a”，结果为 a 的地址，再进行 * 的运算，即 &a 所指向的变量，“ *&a”和“ *p”的作用是一样的，结果都是变量 a。

【例 6.2】指针运算符 * 和 & 的应用实例。

```
int main()
{
    int a, *pa;
    a=100;
```

```
    pa=&a;
    printf("a=%d,*pa=%d\n",a,*pa);
    a=200;
    printf("a=%d,*pa=%d\n",a,*pa);
    *pa=300;
    printf("a=%d,*pa=%d\n",a,*pa);
    return 0;
}
```

程序运行结果如图 6-1-4 所示。

```
a=100,*pa=100
a=200,*pa=200
a=300,*pa=300

--------------------------------
Process exited with return value 0
Press any key to continue . . .
```

图 6-1-4 例 6.2 程序运行结果

【例 6.3】利用指针变量运算的简单应用实例。

```
#include<stdio.h>
int main()
{
    int a=10,b=20,s,t,*pa,*pb;
    pa=&a;
    pb=&b;
    s=*pa+*pb;
    t=*pa**pb;
    printf("a=%d\nb=%d\na+b=%d\na*b=%d\n",a,b,a+b,a*b);
    printf("s=%d\nt=%d\n",s,t);
    return 0;
}
```

程序运行结果如图 6-1-5 所示。

```
a=10
b=20
a+b=30
a*b=200
s=30
t=200

--------------------------------
Process exited with return value 0
Press any key to continue . . .
```

图 6-1-5 例 6.3 程序运行结果

程序中首先定义了整型指针变量 pa、pb，给指针变量 pa 赋值，pa 指向变量 a，给指针变量 pb 赋值，pb 指向变量 b。s= * pa+ * pb；是求 a+b 之和（ * pa 就是 a， * pb 就是 b）。t= * pa * * pb；是求 a * b 之积。最后将结果输出。

2. 指针变量的赋值运算

指针变量的赋值运算有以下几种形式：

（1）指针变量初始化赋值，前面已作介绍。

（2）把一个变量的地址赋予指向相同数据类型的指针变量，如图 6-1-6 所示。

如：

```
int a, * pa;
pa=&a;     /* 把整型变量 a 的地址赋予整型指针变量 pa   */
```

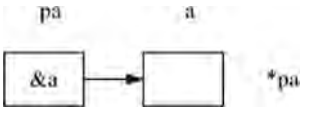


图 6-1-6　变量 a 的地址赋予指针变量 pa

（3）把一个指针变量的值赋予指向相同类型变量的另一个指针变量，如图 6-1-7 所示。

如：

```
int a, * pa=&a, * pb;
pb=pa;     /* 把 a 的地址赋予指针变量 pb */
```

由于 pa、pb 均为指向整型变量的指针变量，因此可以相互赋值。

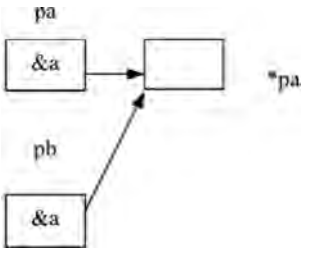


图 6-1-7　指针变量的值赋予另一个指针变量

下面通过一个例子来了解指针变量的应用。

【例 6.4】输入 a 和 b 两个整数，按从大到小的次序输出。

```
#include<stdio. h>
int main( )
{
    int a,b, * pa, * pb, * p;
    pa=&a;
    pb=&b;
    scanf("%d,%d",&a,&b);
    printf("a=%d,b=%d\n",a,b);
```

```
    printf("*pa=%d,*pb=%d\n",*pa,*pb);
    if(a<b)
    {
        p=pa;
        pa=pb;
        pb=p;
    }
    printf("a=%d,b=%d\n",a,b);
    printf("*pa=%d,*pb=%d\n",*pa,*pb);
    return 0;
}
```

若输入为 10，20，则程序运行结果如图 6-1-8 所示。

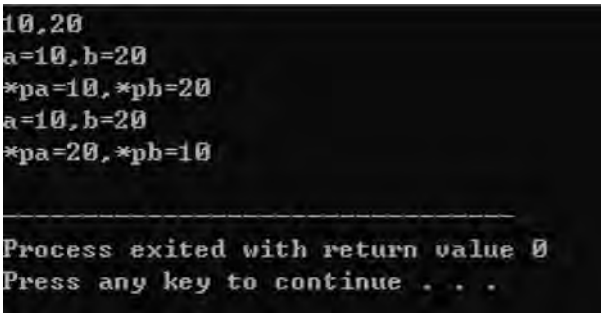


图 6-1-8　例 6.4 程序运行结果

可见，a 和 b 并没有交换，它们还是保持原来的数据不变，但指针变量 pa 和 pb 的值改变了。运行前，pa 原来指向 a，pb 指向 b；运行后，pa 和 pb 的值互换，而 a 和 b 不变。这样在输出 *pa 和 *pb 时，实际输出的是变量 b 和 a 的值，所以输出时先输出 b 的值 20，再输出 a 的值 10。

如果把程序改写成以下这样：

```
#include<stdio.h>
int main()
{
    int a,b,*pa,*pb,t;
    pa=&a;
    pb=&b;
    scanf("%d,%d",&a,&b);
    printf("a=%d,b=%d\n",a,b);
    printf("*pa=%d,*pb=%d\n",*pa,*pb);
    if(a<b)
    {
        t=*pa;
        *pa=*pb;
```

```
        *pb=t;
    }
    printf("a=%d,b=%d\n",a,b);
    printf("*pa=%d,*pb=%d\n",*pa,*pb);
    return 0;
}
```

同样输入 10，20，程序结果如图 6-1-9 所示。

```
10,20
a=10,b=20
*pa=10,*pb=20
a=20,b=10
*pa=20,*pb=10

--------------------------------
Process exited with return value 0
Press any key to continue . . .
```

图 6-1-9　例 6.4 改写后程序运行结果

可见，a 和 b 交换了。运行前，pa 指向 a，pb 指向 b，*pa 和 *pb 的值分别为 10 和 20；在运行后，指针变量 pa 和 pb 所指向的变量的值互换，即把 a 和 b 的值互换了，使得 a 为 20，b 为 10。这样在输出 *pa 和 *pb 时，实际输出的是变量 a 和 b 的值，所以输出时先输出 a 的值 20，再输出 b 的值 10。

3. 指针变量的算术运算

假设指针变量 p1 和 p2 都指向相同类型，有整型变量或常量 n，则可以进行如下算术运算。

若有如下定义语句：

```
float x,y,*p1,*p2;
p1=&x;
p2=&y;
```

（1）自加自减运算 p1++、p1--。

p1++的作用是将指针变量的值指向相邻的下一个同类型单元。按以上的定义，则可知执行 p1++后 p1 值加 4，指向下一个类型为 float 的变量。p1++所代表的地址实际上是 p1+1*d，d 是 p1 所指向数据的类型所占的字节数（对于整型 d=4，对于单精度实型 d=4，对于双精度实型 d=8，对于字符型 d=1）。

同理，p1--的作用是将指针变量的值指向相邻的上一个同类型单元。p1--所代表的地址实际上是 p1-1*d，d 是 p1 所指向数据的类型所占的字节数。

（*p1）++相当于 x++，括号不可省略，若省略了括号，就成了 *p1++，其优先级别相同则按自右至左的结合方向，相当于 *（p1++），指针变量先指向相邻的下一个同类型单元，然后取其值，p1 则不再指向 x。

（2）加减整型量 p1+n、p1-n。

p1+n 的作用是将指针变量的值指向向下第 n 个同类型单元。p1+n 所代表的地址实际上是 p1+n * d（d 的取值同上）。

p1-n 的作用是将指针变量的值指向向上第 n 个同类型单元。p1-n 所代表的地址实际上是 p1-n * d（d 的取值同上）。

（3）同类型指针相减。

C 语言允许两个指向相同类型的指针变量相减，其结果是两指针变量所指向的两地址间同类型的数据的个数。实际上是两个指针值（地址）相减之差再除以该数组元素的长度（字节数）。

4. 指针变量的关系运算

两个指向相同类型的指针变量也可进行关系运算，可支持>、<、>=、<=、==和!=六种运算，其运算结果可以反映两指针变量所指向单元在内存中的分配地址之间的关系。若 p1<p2 成立，则 p1 所指向单元在 p2 所指向单元之前；若 p1>p2 成立，则 p1 所指向单元在 p2 所指向单元之后；若 p1==p2 成立，则 p1 和 p2 指向同一存储单元。

指针变量还可以与 0 比较。设 p 为指针变量，则 p==0 表明 p 是空指针，它不指向任何变量；p!=0 表示 p 不是空指针。空指针是由对指针变量赋予 0 值而得到的。

例如：

```
#define NULL 0
int *p= NULL;
```

对指针变量赋 0 值和不赋值是不同的。指针变量未赋值时，可以是任意值，是不能使用的，否则将造成意外错误。而指针变量赋 0 值后，则可以使用，只是它不指向具体的变量而已。

6.1.3 指针变量作为函数参数

指针变量的一个最常用的功能是作为函数的参数传递到函数中，为函数提供修改调用变量值的方法。在 C 语言中，参数的传递是“按值传送”的，因此，函数不能直接修改所调用函数的变量的值，只有通过传递变量的地址给函数，才能改变该变量的值。

【例 6.5】 题目同上例，输入 a 和 b 两个整数，按从大到小的次序输出。

本例用指针变量作为函数的参数。

```
#include<stdio.h>
int swap(int *pa,int *pb)
{
    int t;
    t=*pa;
    *pa=*pb;
    *pb=t;
}
int main()
{
    int a,b,*p1,*p2,t;
```

```
    p1=&a;
    p2=&b;
    scanf("%d,%d",&a,&b);
    printf("a=%d,b=%d\n",a,b);
    printf("*p1=%d,*p2=%d\n",*p1,*p2);
    if(a<b)
    swap(p1,p2);
    printf("a=%d,b=%d\n",a,b);
    printf("*p1=%d,*p2=%d\n",*p1,*p2);
    return 0;
}
```

同样输入 10，20，程序运行结果如图 6-1-10 所示。

```
10,20
a=10,b=20
*p1=10,*p2=20
a=20,b=10
*p1=20,*p2=10

--------------------------------
Process exited with return value 0
Press any key to continue . . .
```

图 6-1-10　例 6.5 程序运行结果

例 6.5 利用 swap（）函数交换两个变量的值。当 main（）函数执行到语句“if（a<b）swap（p1，p2）;”时，由于条件成立，因此调用了 swap（）函数，在调用时，把指针变量 p1 和 p2 的值传送到形参变量 pa 和 pb 中，传递参数的方式依然是“值传递”。通过传递，形参变量 pa 和 pb 分别获得了 a 和 b 的地址，即指针变量 pa 指向变量 a，指针变量 pb 指向 b。在执行 swap（）函数的函数体时，使 pa 所指向变量的值和 pb 所指向变量的值互换，也就是实现了 a 和 b 的值的互换。

若把程序如下改写：

```
#include<stdio.h>
int swap(int *pa,int *pb)
{
    int *pt;
    pt=pa;
    pa=pb;
    pb=pt;
}
int main()
{
    int a,b,*p1,*p2,t;
```

```
    p1=&a;
    p2=&b;
    scanf("%d,%d",&a,&b);
    printf("a=%d,b=%d\n",a,b);
    printf("*p1=%d,*p2=%d\n",*p1,*p2);
    if(a<b)
    swap(p1,p2);
    printf("a=%d,b=%d\n",a,b);
    printf("*p1=%d,*p2=%d\n",*p1,*p2);
    return 0;
}
```

还是输入 10，20，程序运行结果如图 6-1-11 所示。

```
10,20
a=10,b=20
*p1=10,*p2=20
a=10,b=20
*p1=10,*p2=20

--------------------------------
Process exited with return value 0
Press any key to continue . . .
```

图 6-1-11　例 6.5 改写后程序运行结果

此时，会发现变量 a 和 b 的值未互换，且指针变量 p1 和 p2 所指向变量的值也未互换。为什么会出现这样的结果呢？原因在于 C 语言中，实参和形参之间的传递是“值的单向传递”。在 swap（）函数中虽然实现了指针变量 pa 和 pb 的互换（值的单向传递），使得 pa 指向变量 b，pb 指向变量 a，但 main（）函数中的指针变量 p1 和 p2 并未互换，依然分别指向变量 a 和 b。

6.2　指针和数组

指针和数组有着密切的关系，任何由数组下标来实现的操作都能由指针变量来完成。变量是通过地址来进行访问的，可以直接利用变量名进行直接访问，也可以通过指向变量来进行间接访问。而数组同样可以用指针变量来实现访问操作。数组是由一组具有相同名字、相同类型、不同下标的数据所构成的，数组元素连续存放在内存的存储单元之中，也各自有相应的地址，故指针变量也可以指向数组元素，即把某一数组元素的地址存入一个指针变量之中。

6.2.1　指向数组的指针

指针变量指向数组，通常就是获得数组的首地址。设已有整型数组 a［5］和指向整型变量的指针变量 p，若要令指针变量 p 指向整型数组 a，可以有以下两种方式实现：

（1）数组的名字有两个特性，一是用于标识数组，二是数组的首地址的值，故可赋予指向数组的指针变量 p，如下：

```
int a[5],*p;
p=a;
```

或用指针变量初始化的方法，如下：

```
int a[5],*p=a;
```

（2）数组第一个元素的地址也是整个数组的首地址，也可赋予 p，可写为：

```
p=&a[0];
```

当然也可采取初始化赋值的方法：

```
int a[5],*p=&a[0];
```

数组定义必须在被引用之前，故下面的操作是错误的：

```
int *p=&a[0],a[5]。
```

6.2.2 通过指针变量访问数组元素

前面介绍的指针变量的加减算术运算通常是针对指向数组的指针变量进行的，可以加上或减去一个整数 n。设 pa 是指向数组 a 的指针变量，则 pa+n，pa-n，pa++，++pa，pa--，--pa 运算都是合法运算。指针变量加或减一个整数 n 是把指针指向的当前位置（指向某数组元素）向前或向后移动 n 个位置。还要注意，数组指针变量向前或向后移动一个位置和地址加 1 或减 1 在概念上是不同的。数组可以有不同的类型，各种类型的数组元素所占的字节长度是不同的。如指针变量加 1，即向后移动 1 个位置表示指针变量指向下一个数据元素的首地址，而不只是在原地址基础上加 1。指针变量的加减运算通常只能对数组指针变量进行，对指向其他类型变量的指针变量作加减运算是毫无意义的。两个指针变量之间的运算只有指向同一数组的两个指针变量之间才能进行运算，否则运算毫无意义。

根据指针变量的运算规则，如果指针变量 pa 指向了数组的首地址，那么下标为 i 的数组元素 a［i］的地址可以表示为：pa+i。

那么对于数组元素 a［i］的访问可以通过指针变量表示为：*（pa+i）。

例如：

```
int a[5],*pa;
pa=a;     /*pa 指向数组 a,也指向 a[0]*/
pa=pa+2;  /*pa 指向 a[2],即 pa 的值是 &pa[2]*/
*(pa+2)=10;  /*给 pa+2 所指向元素赋值 10,即 a[2]的值为 10*/
```

由于数组名 a 就是数组的首地址，所以 a+i 也是地址，也指向数组元素 a［i］。它的计算方法和 pa+i 相同，它们的实际地址都为 a+i*d。而对数组元素 a［i］的访问，也可以通过这样的方式进行：*（a+i）。

由此可见，在指针变量 pa 指向数组首地址的前提下，对下标为 i 的数组元素的访问有以下 4 种等价的方法：①a［i］；②pa［i］；③*（a+i）；④*（pa+i）。

但使用中一定要把 a 和 pa 区分开来，两者是不同的。a 是数组的首地址，是一个常量，其值不可改变。pa 是指针变量，在被定义后可以获取数组的首地址，在使用过程中也可以被重新赋值。如 pa++，使 pa 指向数组中下一元素，但 a++是错误的，因为常量的值

不允许改变。下列表达式是有效的：

```
int a[10], *p;
p=a;
p++
```

而下列表达式是无效的：

```
a=p;
a++;
p=&a;
```

【例 6.6】通过指针变量输入和输出 a 数组的 10 个元素。

```
#include<stdio.h>
int main()
{
    int a[10],i,*pa;
    pa=a;
    for(i=0;i<10;i++)
    {
        scanf("%d",pa);
        pa++;
    }
    pa=a;
    for(i=0;i<10;i++)
    {
        printf("%4d",*pa);
        pa++;
    }
    printf("\n");
    return 0;
}
```

输入为：1 2 3 4 5 6 7 8 9 0，程序运行结果如图 6-2-1 所示。

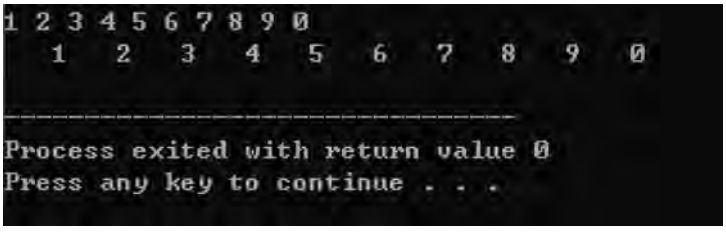


图 6-2-1 例 6.6 程序运行结果

若上述程序改为：

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

```
    int a[10],i,*pa;
    pa=a;
    for(i=0;i<10;i++)
    {
        scanf("%d",pa);
        pa++;
    }
    for(i=0;i<10;i++)
    {
        printf("%4d",*pa);
        pa++;
    }
    printf("\n");
    return 0;
}
```

同样输入1 2 3 4 5 6 7 8 9 0，程序运行结果如图6-2-2所示。

```
1 2 3 4 5 6 7 8 9 0
2293528   1229354419710357002293652419865 3   140018244003512  -1

--------------------------------
Process exited with return value 0
Press any key to continue . . .
```

图6-2-2　例6.6改写后程序运行结果

两个程序的区别只是后一个程序的输出前少了语句“pa=a;”，则输出的显然不是输入的数据，是什么原因引起的错误呢？原因是在输入时，指针变量pa指向的是数组a的首地址，在输入结束后，指针已经指向数组之后，若不新设置pa的初值，则访问的当然不是数组a的元素，结果出现错误。

在使用指针变量对数组进行访问时，一定要关注指针变量的当前值。

【例6.7】利用指针变量求数组的最大值和最小值及其位置。

```
#include<stdio.h>
int main()
{
    int array[10],i,*p,*pmax,*pmin;
    p=array;
    printf("input array a:\n");
    for(i=0;i<10;i++)
    scanf("%d",p++);
    pmax=pmin=array;
    p=array+1;
```

```
    for(i=1;i<10;i++)
    {
        if(*p>*pmax)pmax=p;
        if(*p<*pmin)pmin=p;
        p++;
    }
    printf("数组中的最大值是%d,所处的位置是%d。\n",*pmax,pmax-array);
    printf("数组中的最小值是%d,所处的位置是%d。\n",*pmin,pmin-array);
}
```

程序中首先用指针变量 p 实现数组元素的输入，输入地址的变化由 p++完成。当输入结束时，p 实际指向数组最末元素之后，故在进行下一步操作前，要先将 p 重新指向当前数组。在求解最大值和最小值及其位置时，首先假设第一个元素为最大值和最小值，故将数组首地址存入对应的指针变量 pmax 和 pmin 中，再利用指针变量 p，依次访问数组中其余从 array［0］到 array［9］的元素，和指针变量 pmax、pmin 指向的最大值和最小值进行比较。当指针变量 p 所指向的元素大于指针变量 pmax 时，将 p 的值作为最大值的地址存入 pmax 中，使 pmax 指向最大值；同理，当指针变量 p 所指向的元素小于指针变量 pmin 时，将 p 的值作为最小值的地址存入 pmin 中，使 pmin 指向最小值。循环结束后，pmax 指向最大值，pmax-array 为最大值下标；pmin 指向最小值，pmin-array 为最小值下标。程序运行结果如图 6-2-3 所示。

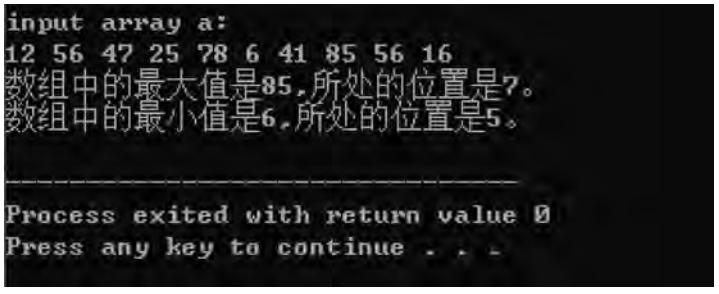


图 6-2-3 例 6.7 程序运行结果

指针变量的使用非常灵活，以下是几个指针变量常用的表达式，必须了解其运算的先后次序。

（1）（*p）++：先取 *p 的值，再将此值加 1 后存入 p 所指向的地址。

（2）++（*p）：先取 *p 的值加 1 存入 p 所指向的地址，再取 *p 的值。

（3）（*p）--：先取 *p 的值，再将此值减 1 后存入 p 所指向的地址。

（4）--（*p）：先取 *p 的值减 1 存入 p 所指向的地址，再取 *p 的值。

（5）*p++：先取 *p 的值，再执行 p++，令 p 指向下一个元素。

（6）*++p：先执行 p++，令 p 指向下一个元素，再取 *p 的值。

（7）*p--：先取 *p 的值，再执行 p--，令 p 指向前一个元素。

（8）*--p：先执行 p--，令 p 指向前一个元素，再取 *p 的值。

学习如下程序，可以进一步了解指针运算：

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

```
int main( )
{
    int n,a[10]={0,1,2,3,4,5,6,7,8,9},*p;
    p=a;
    n=*p++;
    printf("n=%d\n",n);
    n=*++p;
    printf("n=%d\n",n);
    n=++(*p);
    printf("n=%d\n",n);
    n=(*p)++;
    printf("n=%d\n",n);
    return 0;
}
```

程序运行结果如图 6-2-4 所示。

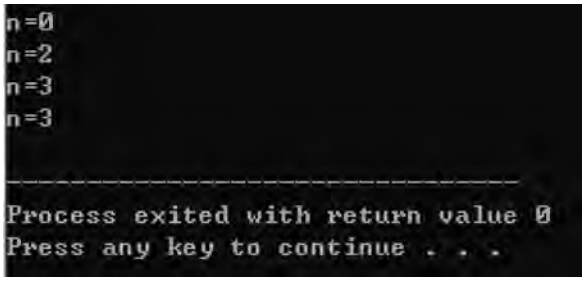


图 6-2-4　指针运算程序运行结果

此程序中共有 4 行输出。第 1 行输出 n 的值为 0。n 被表达式 *p++赋值，由于++在后，所以先取出指针变量 p 所指向的元素的值，即 a［0］的值 0，赋值给 n 后，p 执行++运算，使 p 指向下一元素 a［1］。

第 2 行输出 n 的值为 2。n 被表达式 *++p 赋值，由于++在前，所以 p 先执行++运算，使 p 指向下一元素 a［2］，取出指针变量 p 所指向的元素的值，即 a［2］的值 2，再赋值给 n。

第 3 行输出 n 的值为 3。n 被表达式++（*p）赋值，由于（*p）优先，所以先取出指针变量 p 所指向的元素的值，即 a［2］的值 2，再执行++运算将 a［2］加 1，即 a［2］为 3 后赋值给 n，p 依然指向 a［2］。

第 4 行输出 n 的值为 3。n 被表达式（*p）++赋值，由于++在后，所以先取出指针变量 p 所指向的元素的值，即 a［2］的值 3，赋值给 n 后，执行++运算，使 p 所指向的元素 a［2］加 1，即 a［2］为 4，p 依然指向 a［2］不变。

通常为提高程序的可读性，建议尽可能使用括号，少采用复杂运算。

6.2.3　数组作为函数参数

1. 数组元素地址作为函数参数

如果已定义一个函数：

```
void swap(int x,int y)
{
    int temp;
    temp=x;
    x=y;
    y=temp;
}
```

可知，此函数的作用是将两个参数 x 和 y 的值互换，若进行以下的函数调用：

```
#include<stdio.h>
int main()
{
    int a[10],i,*p=a,*q;
    printf("输入数组 a 的内容:\n");
    for(i=0;i<10;i++)
    {
        scanf("%d",p++);
    }
    p=a;
    q=p+9;
    for(;p<q;p++,q--)
    {
        swap(*p,*q);
    }
    p=a;
    printf("输出数组 a 的内容:\n");
    for(i=0;i<10;i++,p++)
    printf("%3d",*p);
    printf("\n");
    return 0;
}
```

程序运行结果如图 6-2-5 所示，未进行逆序排列。

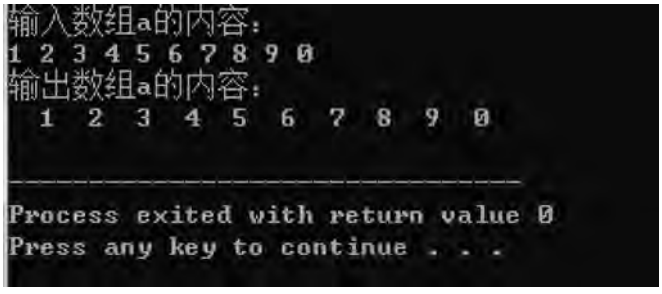


图 6-2-5 调用 swap（）函数程序运行结果

该程序的目的是为了使数组元素输入后，进行函数调用，实现元素的按逆序存放后输出。但运行结果表明数据值未进行逆序排列。原因在于调用函数时使用的 * p，*（p+9-i）就是数组元素，当数组元素作为实参时，情况与变量作为实参时是一样的，是“值的单向传送”方式，调用过程中将 * p，*（p+9-i）的值即数组元素的值单向传送给了形参 x 和 y，当 x 和 y 的值互换时，指针变量 p 和（p+9-i）所指向的元素并不改变。

将程序稍做修改，如下例。

【例 6.8】利用指针实现将数组逆序排列。

```
#include<stdio.h>
void swap(int *x,int *y)
{
    int temp;
    temp=*x;
    *x=*y;
    *y=temp;
}
int main()
{
    int a[10],i,*p=a,*q;
    printf("输入数组 a 的内容:\n");
    for(i=0;i<10;i++)
    {
        scanf("%d",p++);
    }
    p=a;
    q=p+9;
    for(;p<q;p++,q--)
    {
        swap(p,q);
    }
    p=a;
    printf("输出数组 a 的内容:\n");
    for(i=0;i<10;i++,p++)
    printf("%3d",*p);
    printf("\n");
    return 0;
}
```

程序运行结果如图 6-2-6 所示。

该程序实现了数组元素逆序排列的操作，当 main（）函数对 swap（）函数进行调用时，把 p 和 q 作为实参传送到函数的 * x 和 * y 中，即把待互换元素的地址传送到形参中，使函数中的形参的指针变量 x 和 y 也指向数组元素。当指针变量 x 和 y 所指向的对象互换时，也就是对指针变量 p 和 q 所指向的数组元素进行了互换。

图 6-2-6 例 6.8 程序运行结果

C 语言调用函数时参数传递的方法是“值的单向传递”，当用变量名或指针所指向单元作为函数参数时，传递的是变量的值，当用指针作为参数时，传递的是值的地址。除了可以用指针来传递地址外，还可以用数组名来进行地址的传递，下面就来讨论数组名作为函数参数的情况。

2. 数组名作为函数参数

在前面的学习中，已经了解数组的名字有两个特性：一是用于标识数组，二是数组的首地址的值。当用数组名作为函数参数时，由于数组名代表的是数组首地址的值，因此传递的值是地址，在函数定义时，要求对应形参为指针类型。

在用数组名作为函数实参时，相应的形参除了可以是指针变量外，还需要使用形参数组的形式。原因是：在 C 语言中用下标和指针都可以访问一个数组，而下标的表示方法比较直观，更便于理解。在使用中，通常以数组名作为形参，以便与实参数组对应。

数组作为函数参数时，接收数组首地址的函数形参定义有以下几种形式。

（1）函数形参定义为同类型的指针。如：

```
int function(int *p,int n)
{
    …
}
```

在函数的定义中，形参 p 为指针类型，用于接收传递过来的数组首地址；形参 n 用于接收数组的长度，即元素个数。

（2）函数形参定义为有下标数组。如：

```
int function(int a[100])
{
    …
}
```

在此函数的定义中，形参 a 是一个数组长度为 100 的数组名，用于接收实参数组的首地址。

（3）函数形参定义为无下标数组。如：

```
int function(int a[],int n)
{
    …
}
```

在此函数的定义中，形参 a 是一个没有定义数组长度的数组名，数组名代表的是数组

首地址，所以 a 用于接收实参数组的首地址，形参 n 还是用于接收数组的元素个数。

以上 3 种形参的定义形式都可以用于接收实参数组的首地址，3 种形式的形参都是指针类型。在函数调用时，实参可以是数组的首地址，也可以是指向数组的指针变量。

实参数组名代表的是一个指针常量，是一个固定的地址，而形参数组不是一个固定的地址值，只有在函数调用开始时，它才获得值，即实参数组的首地址在函数被执行的过程中，它的值是可以被重新赋予的。例如：

```
func(int a[ ],int n)
{
    int s=0,i;
    for(i=0;i<n;i++)
    {
        s= *a;
        a++;
    }
    return s;
}
```

下面用数组名作为函数参数来完成相关操作。

【例 6.9】将数组 a 中的 n 个元素按逆序存放。

```
#include<stdio.h>
void fun(int x[ ],int n)
{
    int temp,i;
    for(i=0;i<n/2;i++)
    {
        temp=x[i];
        x[i]=x[n-1-i];
        x[n-1-i]=temp;
    }
}
int main( )
{
    int a[10],i;
    printf("输入数组 a 的内容:\n");
    for(i=0;i<10;i++)
    {
        scanf("%d",&a[i]);
    }
    fun(a,10);
    printf("输出数组 a 的内容:\n");
    for(i=0;i<10;i++)
    printf("%3d",a[i]);
```

```
    printf("\n");
    return 0;
}
```

程序运行结果如图 6-2-7 所示。

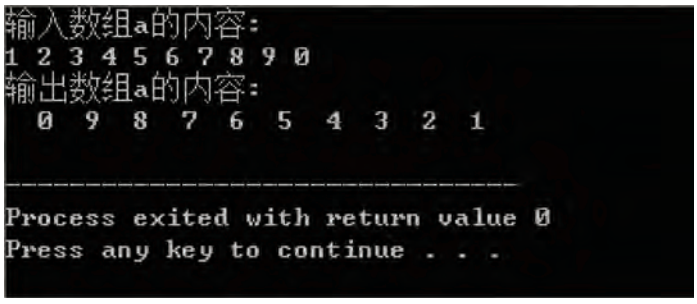


图 6-2-7 例 6.9 程序运行结果

主函数中数组名为 a，通过 scanf（）函数依次输入数组各元素的值。函数 fun（）中的形参数组名为 x，在定义时，若数组长度确定，则可以定义数组个数，若不确定，则可将数组元素个数作为一个形参。在调用时进行传递，两种定义的效果相同。后者的调用更灵活，若函数调用语句“fun（a，10）;”，则表示对数组 a 的前 10 个元素按逆序排列，若函数调用语句“fun（a，6）;”，则表示对数组 a 的前 6 个元素按逆序排列，其余的不变。

还可以把程序稍作改动，把函数 fun（）中的形参设置成指针类型，则运行结果与前一程序相同。程序如下：

```
#include<stdio.h>
void fun(int *p,int n)
{
    int temp,i,*q;
    q=p+n-1;
    for(;p<q;p++,q--)
    {
        temp=*p;
        *p=*q;
        *q=temp;
    }
}
int main()
{
    int a[10],i;
    for(i=0;i<10;i++)
    scanf("%d",&a[i]);
    fun(a,10);
    for(i=0;i<10;i++)
    printf("%3d",a[i]);
    printf("\n");
```

```
    return 0;
}
```

无论传递数组首地址的函数参数用数组名还是指针，实质都是指针，在函数被调用时，该指针通过参数传递均指向数组，在函数中，可以通过指针变量来访问到数组中的各元素，如 *（p+i）、p［i］等方式。若有一个实参数组，希望在函数中改变数组的元素的值，实参和形参的对应关系有以下 4 种。

（1）形参和实参都是数组名，如：

```
int main( )
{
    int a[10];
    ...
    f(a,10);
    ...
    return 0;
}
f(int x[ ],int n)
{
    ...
}
```

在调用时，形参数组和实参数组共用一段内存单元，如图 6-2-8 所示。例 6.9 中的第一个程序就属此类。

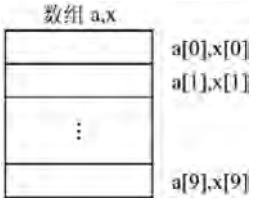


图 6-2-8　形参和实参都是数组名

（2）形参为指针变量，实参为数组名，如：

```
int main( )
{
    int a[10];
    ...
    f(a,10);
    ...
    return 0;
}
f(int *x,int n)
{
    ...
}
```

实参 a 是数组名，形参 x 是指向整型类型的指针变量，在调用时，x 获得 a［0］的地址，即指向 a 数组，如图 6-2-9 所示。随着 x 值的变化，可以访问数组 a 中的每一个元素。例 6.9 中的第二个程序就属此类。

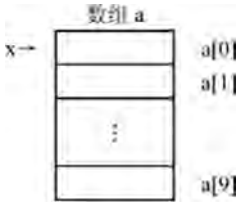


图 6-2-9 形参为指针变量，实参为数组名

（3）形参和实参都为指针变量，如：

```
int main( )
{
    int a[10], *p=a;
    ...
    f(p,10);
    ...
    return 0;
}
f(int *x,int n)
{
    ...
}
```

实参 p 和形参 x 均为指针变量，在 main（）函数中令指针变量 p 指向数组 a，即获得 a［0］的地址，在调用时，x 获得指针变量 p 的值，也就是 a［0］的地址，故指针变量 x 也指向 a 数组，如图 6-2-10 所示。也可以把例 6.9 中的程序改写成如下形式：

```
#include<stdio.h>
void fun(int *p,int n)
{
    int temp,i,*q;
    q=p+n-1;
    for(;p<q;p++,q--)
    {
        temp=*p;
        *p=*q;
        *q=temp;
    }
}
int main( )
{
```

```
    int a[10],i,*ap=a;
    for(i=0;i<10;i++,ap++)
    scanf("%d",ap);
    ap=a;
    fun(ap,10);
    for(i=0;i<10;i++,ap++)
    printf("%3d",*ap);
    printf("\n");
    return 0;
}
```

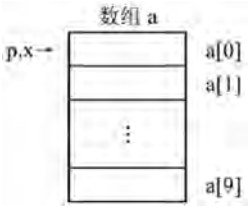


图 6-2-10　形参和实参都为指针变量

（4）形参为数组名，而实参为指针变量，如：

```
int main()
{
    int a[10],*p=a;
    ...
    f(p,10);
    ...
    return 0;
}
f(int x[],int n)
{
    ...
}
```

实参 p 为指针变量，在 main（）函数中令指针变量 p 指向数组 a，即获得 a［0］的地址，在调用时，形参数组 x 获得指针变量 p 的值，也就是 a［0］的地址，故数组 x 和数组 a 共用一段内存单元，如图 6-2-11 所示。也可以把例 6.9 中的程序改写成如下形式：

```
#include<stdio.h>
void fun(int x[],int n)
{
    int temp,i;
    for(i=0;i<n/2;i++)
    {
        temp=x[i];
        x[i]=x[n-1-i];
```

```
        x[n-1-i]=temp;
    }
}
int main()
{
    int a[10],i,*ap=a;
    for(i=0;i<10;i++,ap++)
    scanf("%d",ap);
    ap=a;
    fun(ap,10);
    ap=a;
    for(i=0;i<10;i++,ap++)
    printf("%3d",*ap);
    printf("\n");
    return 0;
}
```

图 6-2-11 形参为数组名，而实参为指针变量

以上 4 种方法实际上都是使用指针变量进行地址的传递。

但应注意，如果用指针变量作为实参时，必须先给指针变量进行赋值，使其指向一个已定义的单元。切记不可出现如下使用方法：

```
int main()
{
    int *p;
    ...
    f(p,10);
    ...
    return 0;
}
f(int x[],int n)
{
    ...
}
```

程序出错，原因是指针变量 p 没有确定的值，无法对其指向变量进行操作。

【例 6.10】用选择法对 n 个整数按升序排列。

先令实参为指针变量，形参为数组名，程序如下：

```
#include<stdio. h>
void sort( int x[ ] ,int n)
{
    int i,j,t;
    for( i=0;i<n-1;i++)
    {
        for( j=i+1;j<n;j++)
        if( x[ i]>x[ j]){
        t=x[ i];
        x[ i]=x[ j];
        x[ j]=t;
        }
    }
}
int main( )
{
    int a[ 10] ,i, * p;
    p=a;
    printf( "请输入 10 个整数:\n" );
    for( i=0;i<10;i++,p++)
    scanf( "%d" ,p);
    p=a;
    sort( p,10);
    for( i=0;i<10;i++,p++)
    printf( "%3d" , * p);
    printf( "\n" );
    return 0;
}
```

程序运行结果如图 6-2-12 所示。

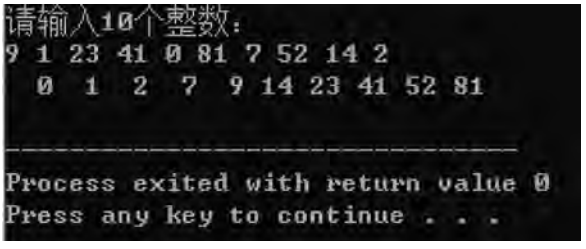


图 6-2-12　例 6.10 程序运行结果

还可以保持主函数不变，只修改 sort（）函数，使形参为指针变量的形式来完成，修改后的 sort（）函数程序如下：

```
void sort( int  * x,int n)
{
```

```
    int i,j,t;
    for(i=0;i<n-1;i++)
    {
        for(j=i+1;j<n;j++)
        if( *(x+i)>*(x+j)){
        t= *(x+i);
         *(x+i)= *(x+j);
         *(x+j)=t;
        }
    }
}
```

运行结果与例 6.10 相同，均可完成数组元素按升序排列。

6.2.4 指向多维数组的指针

和一维数组一样，多维数组也可以用指针指向多维数组中的元素，但在指向多维数组的指针变量的定义和使用上会复杂一些。

1. 多维数组地址的表示方法

设有整型二维数组 a［3］［4］：

```
int a[3][4]={{0,1,2,3},{4,5,6,7},{8,9,10,11}};
```

设数组 a 的首地址为 1000，则各下标变量的首地址及其值如图 6-2-13 所示。

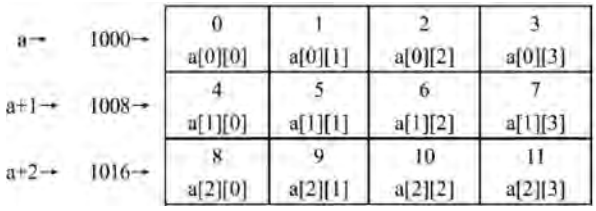


图 6-2-13 二维数组地址与指针关系示意图

C 语言允许把一个二维数组分解为多个一维数组来处理。故数组 a［3］［4］可分解为 3 个一维数组，分别是 a［0］，a［1］，a［2］。每一个一维数组又含有 4 个元素。例如 a［0］数组，含有 a［0］［0］，a［0］［1］，a［0］［2］，a［0］［3］4 个元素。数组及数组元素的地址表示如下：a 是二维数组名，也是二维数组 0 行的首地址，等于 1000。a［0］是第一个一维数组的数组名和首地址，因此也为 1000。*（a+0）或 * a 是与 a［0］等效的，它表示一维数组 a［0］的 0 号元素的首地址，也为 1000。&a［0］［0］是二维数组 a 的 0 行 0 列元素首地址，同样是 1000。因此，a，a［0］，*（a+0），* a，&a［0］［0］是相等的。同理，a+1 是二维数组 1 行的首地址，等于 1008。a［1］是第二个一维数组的数组名和首地址，因此也为 1008。&a［1］［0］是二维数组 a 的 1 行 0 列元素地址，也是 1008。因此 a+1，a［1］，*（a+1），&a［1］［0］是等同的。由此可得出：a+i，a［i］，*（a+i），&a［i］［0］是等同的。此外，&a［i］和 a［i］也是等同的。因为在二维数组中不能把 &a［i］理解为元素 a［i］的地址，不存在元素 a［i］。

由此得出：a［i］，&a［i］，*（a+i）和 a+i 也都是等同的。另外，a［0］也可以看

成 a［0］+0，是一维数组 a［0］的 0 号元素的首地址，而 a［0］+1 则是 a［0］的 1 号元素首地址，由此可得出 a［i］+j 是一维数组 a［i］的 j 号元素首地址，它等于 &a［i］［j］。由 a［i］=*（a+i）得 a［i］+j=*（a+i）+j，由于*（a+i）+j 是二维数组 a 的 i 行 j 列元素的首地址，该元素的值等于*（*（a+i）+j）。

【例 6.11】输入 3 行 4 列的整型数组，求所有元素的和。

程序如下：

```
#include<stdio.h>
int main()
{
    int a[3][4],i,j,sum=0;
    for(i=0;i<3;i++)
    for(j=0;j<4;j++)
    {
        scanf("%d",a[i]+j);
        sum=sum+*(*(a+i)+j);
    }
    printf("sum=%d\n",sum);
    return 0;
}
```

程序运行结果如图 6-2-14 所示。

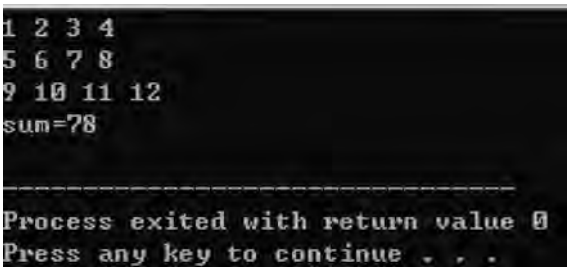


图 6-2-14　例 6.11 程序运行结果

2. 多维数组的指针变量

若有以下定义语句：

```
int a[3][4],*p=a;
```

则指针变量 p 指向数组 a，但指针变量 p 和数组名 a 的性质不同。除了数组名 a 为常量、指针变量 p 是变量外，在进行算术运算时也不同。数组名 a 加 1 时，地址下移一行；指针变量 p 加 1 时，地址指针下移一个元素。所以，若用指针变量 p 和行下标 i、列下标 j 来表示数组元素 a［i］［j］，可以通过表达式计算出地址，然后再通过间接方式访问，形式为：

```
*(数组首地址+行下标*列数+列下标)
```

在以上定义的前提下，若要用指针变量 p 来访问数组元素 a［i］［j］，则可表示为：

```
*(p+i*4+j)
```

其中 4 就是每行所包含的列数。

由于可以把二维数组 a 分解为一维数组 a［0］，a［1］，a［2］，那么设 p 为指向二维数组的指针变量，可定义为：

```
int(*p)[4];
```

它表示 p 是一个指针变量，它指向二维数组 a 或指向第一个一维数组 a［0］，其值等于 a，a［0］或 &a［0］［0］等，而 p+i 则指向一维数组 a［i］。从前面的分析可得出 *（p+i）+j 是二维数组第 i 行 j 列的元素的地址，而 *（*（p+i）+j）则是 i 行 j 列元素的值。

二维数组指针变量说明的一般形式为：

```
类型说明符(*指针变量名)[长度]
```

其中“类型说明符”为所指数组的数据类型。“*”表示其后的变量是指针类型。“长度”表示二维数组分解为多个一维数组时，一维数组的长度，也就是二维数组的列数。应注意“（*指针变量名）”两边的括号不可少，如缺少括号则表示是指针数组，意义就完全不同了。

【例 6.12】键盘输入待输出元素的行下标和列下标，输出指定元素的值。

程序如下：

```
#include<stdio.h>
int main()
{
    int a[3][4]={{2,4,6,8},{10,12,14,16},{18,20,22,24}};
    int(*p)[4],i,j;
    p=a;
    printf("i=");
    scanf("%d",&i);
    printf("j=");
    scanf("%d",&j);
    printf("a[%d][%d]=%d\n",i,j,*(*(p+i)+j));
    return 0;
}
```

程序运行结果如图 6-2-15 所示。

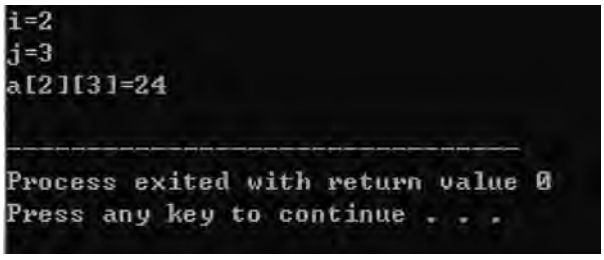


图 6-2-15　例 6.12 程序运行结果

3. 指向多维数组的指针变量作为函数

与一维数组一样，多维数组名也可以作为函数参数传递。

设调用函数说明了如下的一个二维数组：

```
int a[3][4];
```

该数组是 3 行 4 列的二维整型数组，被调用函数参数可以说明为如下 3 种形式。

（1）参数为行列数全部说明的二维整型数组形式。

```
int fun(int x[3][4])
{
    ...
}
```

其中，形参数组 x 的行和列的数量都是确定的。

（2）参数为省略行数的二维整型数组形式。

```
int fun(int x[][4])
{
    ...
}
```

其中，形参数组 x 的列的数量是确定的，但行的数量是不确定的。

（3）参数为行列数全部说明的二维整型数组形式。

```
int fun(int( *x)[4])
{
    ...
}
```

其中，(*x) [4] 的含义是：x 是一个指向有 4 个元素的一维数组的指针，x 加 1 时，地址移动 4 个元素。

【例 6.13】 有 3 个学生，各学 4 门课程，计算总平均分数，以及第 n 个学生的成绩。

程序如下：

```
#include<stdio.h>
void average(float *p,int n)
{
    float *q;
    float sum=0,aver;
    q=p+n-1;
    for(;p<=q;p++)
    sum=sum+*p;
    aver=sum/n;
    printf("average=%6.2f\n",aver);
}
void search(float(*p)[4],int n)
{
    int i;
    printf("The score of No.%d are:\n",n);
```

```
    for(i=0;i<4;i++)
    printf("%7.2f\n",*(*(p+n)+i));
}
int main()
{
    float score[3][4]={{89,78,69,75},{91,66,52,78},{87,73,78,66}};
    average(*score,12);
    search(score,2);
    return 0;
}
```

程序运行结果如图 6-2-16 所示。

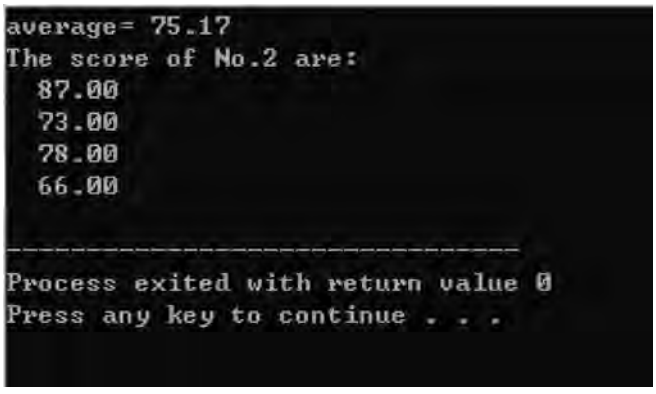


图 6-2-16 例 6.13 程序运行结果

6.3 指针和字符串

字符串在内存中以两种存放形式出现，一种是前面了解的字符串常量，比如"CLanguage"，内容不可改变；另一种是存入字符数组，数组的每个元素存放一个字符，可以对数组元素进行修改。但两种方式均将数据存于内存中，因此都可以用指针方式访问其中数据。

6.3.1 指针对字符串的访问

字符串指针变量的定义和使用与指向字符变量的指针变量说明是相同的。两者按对指针变量的赋值的不同来进行区别。对指向字符变量的指针变量应赋予该字符变量的地址。如：

```
char c,*p=&c;
*p='a';
```

表示 p 是一个指向字符变量 c 的指针变量，可以通过指针变量 p 给变量 c 赋值。

而：

```
char *s="C Language";
```

则表示 s 是一个指向字符串的指针变量，把字符串的首地址赋予 s，也可以写成以下形式：

```
char *s="C Language";
s="C Language";
```

两者等价。

【例 6.14】利用指针实现字符串的访问。

```
#include<stdio.h>
int main()
{
    char *ps;
    ps="how are you! \n";
    printf("%s",ps);
    return 0;
}
```

程序运行结果如图 6-3-1 所示。

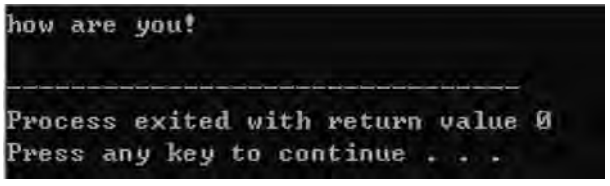


图 6-3-1　例 6.14 程序设计结果

该程序中，首先定义 ps 是一个字符指针变量，然后把字符串的首地址赋予 ps（必须赋值整个字符串，以便编译系统把该串装入连续的一块内存单元）。程序中的 char *ps; ps="C Language"; 等效于 char *ps="C Language";，最后输出字符串中 n 个字符后的所有字符。

【例 6.15】定义一个字符数组并初始化，然后输出该字符串的子串。

```
#include<stdio.h>
int main()
{
    char *ps="I am a girl";
    int n=7;
    printf("%s\n",ps);
    ps=ps+n;
    printf("%s\n",ps);
    return 0;
}
```

程序运行结果如图 6-3-2 所示。

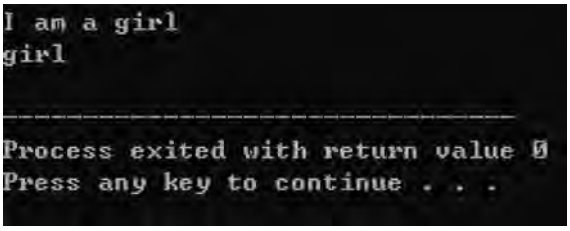


图 6-3-2 例 6.15 程序设计结果

在该程序中对 ps 初始化的同时就把字符串首地址赋予 ps，在 ps=ps+7 之后，ps 指向字符“g”，因此输出为“girl”。

【例 6.16】在输入的字符串中查找有无' d' 字符。

```
#include<stdio.h>
int main()
{
    char st[20],*pd;
    int i;
    printf("input a string:\n");
    pd=st;
    scanf("%s",pd);
    for(i=0;pd[i]!='\0';i++)
    if(pd[i]=='d')
    break;
    if(pd[i]=='\0')
    printf("There is no 'd' in the string.\n");
    else
    printf("There is a 'd' in the string.\n");
    return 0;
}
```

程序运行结果如图 6-3-3 所示。

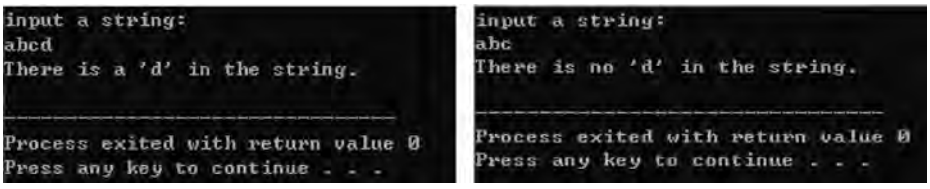


图 6-3-3 例 6.16 程序运行结果

该程序中，利用指向字符数组的指针变量 pd 对 st 数组中的每个元素通过 pd［i］的方式进行访问。在循环中，只要通过下标的变化就可以访问所有元素。循环结束时，可以通过对下标 i 的判断来得到结果。

6.3.2 字符串指针作为函数参数

要将一个字符串从一个函数传递到另一个函数，用的是地址传递的方式。要实现地址传递，可以通过将字符数组名作为参数，或者指向字符数组的指针变量作为参数，这样，在被调用的函数中若改变了字符串内容，则主调函数中相应的字符串内容也随之改变。

【例 6.17】要求对两个字符串进行比较，并且不能使用 strcmp（）函数。函数 cmps（）的形参为字符数组名，如果两个字符串相等，返回值为 0，否则返回值为非 0。

```
#include<stdio.h>
char cmps(char x[],char y[])
{
    int i;
    for(i=0;x[i]!='\0';i++)
    if(x[i]!=y[i])break;
    return(x[i]-y[i]);
}
int main()
{
    int f;
    char a[20],b[20];
    printf("please input a:");
    gets(a);
    printf("please input b:");
    gets(b);
    f=cmps(a,b);
    if(f==0)
    printf("a string and b string equal.\n");
    else
    printf("a string and b string does not equal.\n");
    return 0;
}
```

程序运行结果图 6-3-4 所示。

图 6-3-4 例 6.17 程序运行结果

该程序中，通过对两个数组中相应位置的字符进行判断，即对数组 x 和数组 y 中的字符自左至右逐个比较，直到出现不同字符或者 x 数组遇到‘\0’为止。最终将最后一次比较的值进行返回。在主函数中，以数组名 a、b 为实参，调用 cmps（）函数。由于采用数组名进行参数传递，故数组 a 和 x、数组 b 和 y 均指向同一数组，在主函数和 cmps（）函数

中均可使用这些字符串。

【例 6.18】要求把一个字符串的内容复制到另一个字符串中，并且不能使用 strcpy（）函数。函数 cpys（）的形参为两个字符指针变量。指针变量 pa 指向源字符串，指针变量 pb 指向目标字符串。

```
#include<stdio.h>
void cpys(char *pa,char *pb)
{
    while((*pb=*pa)!='\0')
    {
        pb++;
        pa++;
    }
}
int main()
{
    char *p1="CHINA";
    char b[10];
    char *p2;
    p2=b;
    cpys(p1,p2);
    printf("string a=%s\nstring b=%s\n",p1,p2);
    return 0;
}
```

程序运行结果如图 6-3-5 所示。

```
string a=CHINA
string b=CHINA
Press any key to continue
```

图 6-3-5 例 6.18 程序运行结果

该程序完成了两项工作：一是把 pa 指向的源字符复制到 pb 所指向的目标字符中，二是判断所复制的字符是否为'\0'，若是'\0'则表明源字符串结束，不再循环。否则，pb 和 pa 都加 1，指向下一字符。在主函数中，以指针变量 p1、p2 为实参，分别取得确定值后调用 cpys（）函数。由于采用的指针变量 p1 和 pa、p2 和 pb 均指向同一字符串，因此在主函数和 cpys（）函数中均可使用这些字符串。

也可以把 cpys（）函数简化为以下形式：

```
void cpys(char *pa,char *pb)
{
    while((*pb++=*pa++)!='\0');
}
```

即把指针的移动和赋值合并在一个语句中。进一步分析还可发现‘\0’的ASCII码为0，对于while语句只看表达式的值为非0就循环，为0则结束循环，因此也可省去“!='\0'”这一判断部分，而写为以下形式：

```
void cpys(char *pa,char *pb)
{
    while(*pb++=*pa++);
}
```

表达式的意义可解释为，源字符向目标字符赋值，移动指针，若所赋值为非0则循环，否则结束循环。这样使程序更加简捷。

6.3.3 使用字符串指针变量与字符数组的区别

用字符数组和字符指针变量都可实现字符串的存储和运算，但是两者是有区别的。在使用时应注意以下几个问题。

（1）字符串指针变量本身是一个变量，用于存放字符串的首地址。而字符串本身是存放在以该首地址为首的一块连续的内存空间中并以‘\0’作为串的结束。字符数组是由若干个数组元素组成的，它可用来存放整个字符串。

（2）对字符串指针赋值方式：

```
char *ps="C Language";
```

可以写为：

```
char *ps;
ps="C Language";
```

而对数组方式：

```
char st[]={"C Language"};
```

不能写为：

```
char st[20];
st={"C Language"};
```

只能对字符数组的各元素逐个赋值。

（3）如果已经定义了一个字符数组，在编译时，它有确定的地址，并被分配指定长度的内存单元。而对于一个已定义的字符指针变量，系统给指针变量分配内存单元，此内存单元用于存放一个字符变量的地址，即令指针变量指向一个字符数据，但若没有对指针变量赋值的话，则该指针变量不指向一个确定的字符。前面说过，在一个指针变量未取得确定地址前使用是危险的，容易引起错误。例如：

```
char st[20];
scanf("%s",str);
```

是可以的。若改成以下形式：

```
char *ps;
scanf("%s",ps);
```

由于指针变量ps未赋值，故ps中的值不确定，可能指向的是内存中空白的用户存储区，执行scanf("%s",ps);后能正常运行，也可能指向已存放指令或者数据的有用内存，一旦数据读入，就会破坏程序，甚至破坏系统，产生严重的后果。因此通常采用以下

形式：

```
char *ps,str[20];
ps=str;
scanf("%s",ps);
```

这样，指针变量 ps 先获得一个确定的值，即令 ps 指向字符数组 str 的第一个元素，然后输入一个字符串，把该字符串存放在该地址开始的连续存储单元中。

从以上几点可以看出字符串指针变量与字符数组在使用时的区别，同时也可发现使用指针变量更加方便。

6.4 指针数组和指向指针的指针

数组指针只是一个指针变量，似乎是 C 语言里专门用来指向二维数组的，它占有内存中一个指针的存储空间。指针数组是多个指针变量，以数组形式存在内存当中，占有多个指针的存储空间。而指向指针的指针声明的是指针，只是这个指针指向另一个指针。

6.4.1 指针数组

同类型的变量可以构造成数组，指针变量也是变量，同类型的指针变量也可以构造为数组，这就是指针数组。这样，指针数组中的每一个元素都相当于一个指针变量。一维指针数组的定义格式如下：

```
类型 *指针数组名[数组长度];
```

类型是定义指针数组的每一个元素所指向对象的类型；指针数组名前的“*”是指针标志；指针数组名与指针名一样，是数组的标识，用于访问数组元素，它的值是指针数组在内存中的首地址；数组长度用于确定数组元素的个数。例如：

```
int *p[10];
```

该语句中［］比*级别高，因此先形成 p［10］，即定义了一个由 10 个元素组成的数组，然后再与前面的*结合，表示此数组的元素为指针类型，且每个元素都指向一个整型变量。

但注意不要写成“int（*p）［10］;”的形式，该语句表示指向由 10 个元素组成的一维数组的指针变量。

【例 6.19】 统计 3 行 4 列整型二维数组中偶数的个数，并求出偶数的和，输出结果。

```
#include<stdio.h>
int main()
{
    int a[3][4];
    int *p[3];
    int i,j,count,sum;
    printf("input a:\n");
    for(i=0;i<3;i++)
    {
        p[i]=a[i];
        for(j=0;j<4;j++)
```

```
        scanf("%d",p[i]+j);
    }
    count=0;
    sum=0;
    for(i=0;i<3;i++)
    for(j=0;j<4;j++)
    if(p[i][j]%2==0)
    {
        sum=sum+p[i][j];
        count++;
    }
    printf("the sum is %d\n",sum);
    printf("the count is %d\n",count);
    return 0;
}
```

程序运行结果如图 6-4-1 所示。

```
input a:
1 2 3 4
5 6 7 8
9 10 11 12
the sum is 42
the count is 6

--------------------------------
Process exited with return value 0
Press any key to continue . . .
```

图 6-4-1　例 6.19 程序运行结果

6.4.2　指向指针的指针

指向指针的指针变量中存放的是另一指针变量的地址，故称其为指向指针的指针变量。该指针变量可以通过多重间接方式访问内存中的数据。

指向指针的指针变量的实质也是内存的变量，在使用时也必须遵循“先定义，后使用”的原则，应先定义该指针变量，再使其指向某一指针变量后通过指针来访问对象，在访问中需要进行两次间接运算。

指向指针的指针变量的定义的一般形式为：

```
类型说明符 **变量名;
```

语句中，“类型说明符”是变量所指向的指针指向的数据类型，“**”是指向指针的指针变量的标志。

【例 6.20】指向指针的指针变量的简单应用。

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

```
{
    int i, *p, **pp;
    i=10;
    p=&i;
    pp=&p;
    printf("i=%d\n*p=%d\n**pp=%d\n",i,*p,**pp);
    return 0;
}
```

该程序运行结果如图 3-4-2 所示。程序中通过语句“p=&i;”使指针变量 p 指向变量 i，通过语句“pp=&p;”使指向指针的指针变量 pp 指向指针变量 p，这样 pp 就可以进行对 i 的访问了，访问表达式为：

```
**pp
```

即通过两次间接运算实现。由运算符的结合性可以知道，“*”是从右到左结合，**pp 相当于 *(*pp)，第一级间接运算后得到的结果是 *p，再对第二级间接运算 *p 后得到结果就是 i。

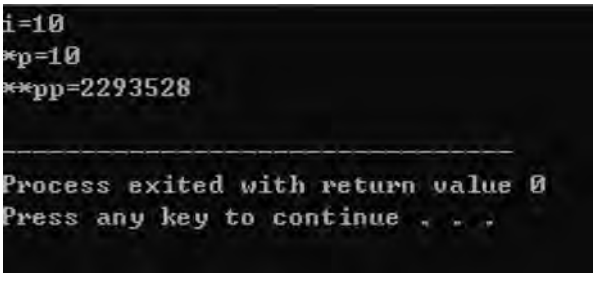


图 6-4-2 例 6.20 程序运行结果

指向指针的指针变量对简单变量进行访问一般很少见，主要用于指针数组以及在函数传递中传递二维数组等。下面举例来了解指向指针的指针变量与指针数组的应用。

【例 6.21】求二维数组的平均值。

```
#include<stdio.h>
int main()
{
    int a[3][4], *p[3], **pp;
    int sum=0,i,j;
    float ave;
    for(i=0;i<3;i++)
    {
        p[i]=a[i];
        for(j=0;j<4;j++)
        scanf("%d",p[i]+j);
    }
```

```
    pp=p;
    for(i=0;i<3;i++)
    for(j=0;j<4;j++)
    sum=sum+*(*(pp+i)+j);
    ave=sum/12.0;
    printf("The average of a is %7.2f\n",ave);
    return 0;
}
```

程序运行结果如图 6-4-3 所示。程序中 p 是指针数组，p 数组中的元素分别指向二维数组的各行，pp 是指向指针的指针变量，语句“pp=p;”使 pp 获得指针数组 p 的地址，即 pp 指向 p 数组，此时，pp+i 是 p［i］的地址，故 *（pp+i）就是 p［i］，也就是 *（*（pp+i）+j）即 *（a［i］+j），所以 *（*（pp+i）+j）就是 *（a［i］+j），也就是 a［i］［j］。

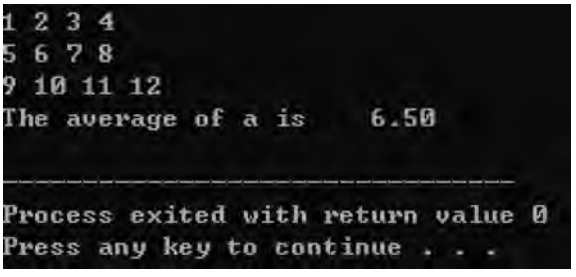


图 6-4-3　例 6.21 程序运行结果

6.4.3　指针的其他用法

1. 指向函数的指针

指针变量可以指向单个变量、数组、字符串，也可以指向函数。函数在被编译时，获得一个入口地址，将此地址存放入一个指针变量，就可以通过该指针变量调用此函数。该指针变量称为指向函数的指针变量。

指向函数的指针变量的使用有 3 个步骤。

（1）定义指向函数的指针变量。

指向函数的指针变量定义的一般形式为：

```
类型说明符(*变量名)();
```

例如：

```
int(*p)();
```

定义了一个指向函数的指针变量 p，且该函数的返回值为 int 类型，即该指针变量可以存放一个返回值类型为 int 类型的函数的入口地址。

（2）将定义后的指针变量指向函数。在指向函数的指针变量定义后，它不是固定指向哪个函数，而只是表示定义了这样一个类型的变量，可以专门用来存放函数的入口地址。程序中把哪个函数的入口地址给它，它就指向哪个函数。在一个程序中，一个指向函数的

指针变量可以指向返回值相同的不同函数。

在给指向函数的指针变量赋值时，只需要给出函数名而不必给出参数。例如：

```
int main()
{
    int(*p)();
    ...
    p=fun
    ...
    return 0;
}
int fun(int a,int b)
{
    return a+b;
}
```

fun 是函数名，是函数的首地址，将 fun 赋值给指向函数的指针变量 p，实质上就是使 p 获得了函数 fun（）的入口地址，即指向了函数 fun（）。

（3）通过指针调用其指向的函数。用指向函数的指针变量调用函数时，只需要将（*p）代替函数名即可，当然（*p）后的括号中的参数根据需要填写。

步骤（2）中，通过指向函数的指针变量调用函数 fun（）的格式为：

```
s=(*p)(x,y);
```

【例 6.22】求 x、y 中的最大值。

```
#include<stdio.h>
int main()
{
    int getmax(int,int);
    int x,y,max;
    int(*p)();
    printf("input x,y:\n");
    scanf("%d,%d",&x,&y);
    p=getmax;
    max=(*p)(x,y);
    printf("x=%d\ny=%d\nmax=%d\n",x,y,max);
    return 0;
}
int getmax(int a,int b)
{
    return(a>b? a:b);
}
```

程序运行结果如图 6-4-4 所示。程序中，语句“int（*p）（）;”定义了一个指向函数的指针变量 p，该函数的返回值为整型。语句“p=getmax;”是将函数 getmax（）的入口地址赋值给 p。语句“max=（*p）（x，y）;”实现了通过指向函数的指针变量对函数进行调用，并把返回值赋值给变量 max。

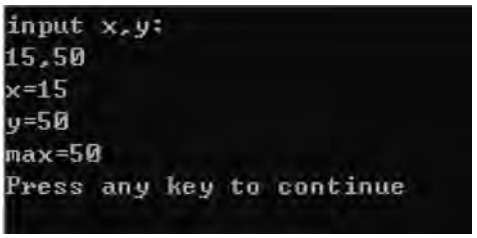


图 6-4-4　例 6.22 程序运行结果

2. 返回指针值的函数

函数的返回值可以是整型值、字符型值、实型值等，在 C 语言中函数的返回值还可以是指针类型的数据，即返回地址。

这种返回指针值的函数定义形式如下：

```
类型说明符 *函数名(参数表列)
{函数体}
```

例如：

```
int *p(int x,int y)
{
    ...
}
```

p 是函数名，一旦对当前函数调用，即可得到一个指向整型数据的指针值。

【例 6.23】通过函数在一字符串中搜索一个特定字符第一次出现的地址，在主函数调用并输出结果。

```
#include<stdio.h>
#include<string.h>
char *search(char *,char);
int main()
{
    char str[80];
    char ch,*pch;
    printf("input string:\n");
    gets(str);
    printf("input a char:\n");
    scanf("%c",&ch);
    pch=search(str,ch);
    if(pch==NULL)
    printf("The char %c not found.\n",ch);
    else
    printf("The position is %d\n",pch-str);
    return 0;
}
```

```
char *search(char *s,char c)
{
    char *p=NULL;
    while(*s!='\0')
    {
        if(*s==c)
        {
            p=s;
            break;
        }
        else
        s++;
    }
    return p;
}
```

程序运行结果如图 6-4-5 所示。

```
input string:
china
input a char:
a
The position is 4

--------------------------------
Process exited with return value 0
Press any key to continue . . .
```

图 6-4-5　例 6.23 程序运行结果

为便于比较和记忆，下面把与指针变量相关的定义列在一起。

（1）int *p;　p 为指向整型数据的指针变量。

（2）int *p[N];　p 为指针数组，该数组由 N 个指向整型数据的指针组成。

（3）int (*p)[N];　p 为指针变量，该指针变量指向含有 N 个元素的整型一维数组。

（4）int *p();　p 为返回一个指针的函数，该指针指向整型数据。

（5）int (*p)();　p 为一个指向函数的指针变量，该函数的返回值类型为整型。

（6）int **p;　p 为一个指向指针的指针变量，它所指向的指针所指向的数据为整型。

指针是 C 语言中最重要的一种数据类型，也是最能体现 C 语言特色的一种数据类型。指针变量的使用提高了程序的效率，并且在函数调用时更加灵活，可以在函数中将多个结果返回到主调函数等。但也正是指针应用的灵活性，使得在指针操作的过程中的错误不易被发现。因此，使用指针务必谨慎，尽可能多上机进行调试，积累指针使用的经验，利用指针这一数据类型编写出更多具有特色的优质程序。

项目三　歌曲信息管理系统

项目设置目的

该项目以歌曲信息管理系统为背景，学习结构体和文件的内容。该项目分解为两个任务。通过本项目的实现，掌握小型系统程序设计的基本方法、基本框架的搭建和模块化程序设计的思想，能够使用结构体变量、结构体数组和函数编写小型的应用程序。

项目分析

本项目实现了对批量歌曲信息的管理。主模块应包含菜单显示模块、录入歌曲信息模块、浏览歌曲列表模块、查询歌曲模块、删除歌曲模块、保存文件模块和读取文件模块，每个模块都定义为一个功能相对独立的函数。本项目设计的知识点主要包括函数、数组、结构体和文件操作等内容。

系统各模块的功能说明如下：

（1）菜单显示模块。

（2）录入歌曲信息模块，输入歌曲名、作词、演唱者、发行日期。歌曲名输入 Q 即可终止录入。

（3）浏览歌曲列表模块，显示录入的歌曲信息。

（4）查询歌曲模块，可根据歌曲名、作词或演唱者来查询歌曲信息。

（5）删除歌曲模块，根据歌曲名、作词或演唱者来删除歌曲信息，并对原有的歌曲列表进行排序。

（6）保存文件模块，将输入的歌曲信息列表保存到 songlist. txt 文件中。

（7）读取文件模块，读取 record. txt 文件中的歌曲列表信息。

在学习完理论知识后，学生可以自行完善系统，实现例如插入歌曲信息、修改歌曲信息、按照演唱者姓名排序等功能。

项目分解

任务七　歌曲信息的添加、浏览和删除

任务八　歌曲数据的存储

任务七 歌曲信息的添加、浏览和删除

学习目标

- 理解和掌握结构体的定义和调用方法；
- 理解和掌握结构体数组的定义和引用；
- 理解共用体和枚举类型的构造、定义和引用。

一、任务描述

本任务将结合歌曲信息管理系统项目添加、浏览和删除歌曲信息，由主函数 main（）、添加歌曲函数 enter（）、删除歌曲函数 del（）等功能模块组成。

二、知识要点

本任务主要涉及结构体类型的定义、结构体变量的定义与引用、结构体与数组的综合运用以及一级结构体数组在函数之间的传递。

三、任务分析

对于单个数据，可以通过定义变量进行存储和处理，对于数目固定、数据类型相同的一组数据，可以通过数组来描述和处理。但在实际中，一组数据往往具有不同的数据类型，用单一的基本数据类型和数组都难以表示，C 语言中的结构体、共用体数据类型能够实现这一功能。

在本任务中通过定义一个结构体数组 struct song s［N］实现对歌曲信息的管理，在调用函数时，将结构体数组名作为参数，但是实际上歌曲数量在 0～N 之间，是随着增加和删除发生变化的。

本项目的主体模块包括菜单显示模块、录入歌曲信息模块、浏览歌曲列表模块、查询歌曲模块、删除歌曲模块、保存文件模块和读取文件模块。本任务中将每个模块都定义为一个功能相对独立的函数，各函数名如下：

（1）菜单显示模块，函数定义为 menu（）。

（2）录入歌曲信息模块，函数定义为 enter（struct song s［］，int n）。

（3）浏览歌曲列表模块，函数定义为 list（struct song s［］，int n）。

（4）查询歌曲模块，函数定义为 search（struct song s［］，int n）。

（5）删除歌曲模块，函数定义为 del（int n）。

（6）保存文件模块，函数定义为 save（int n）。

（7）读取文件模块，函数定义为 read（）。

（3）～（7）的函数中都需要歌曲信息，因此使用数组 score［］作为形参。

四、源代码参考

```
/*==========预处理命令==========*/
#include<stdio.h>
```

```
#include<conio. h>
#include<stdlib. h>
#include<string. h>
#define N 10
/*═══════════════════════函数声明═══════════════════════*/
void menu( );
int enter(struct song s[ ],int n);
void list(struct song s[ ],int n);
void search(struct song s[ ],int n);
int del(int n);
int save(int n);
int read( );
/*----------定义结构体--------*/
struct song
{
    char name[40];   //歌曲名称
    char writer[25];   //作词
    char singer[25];   //演唱者
    char year[ 25];   //发行年份
}s[N],s1,s2,s3;
/*----------main( )主函数--------*/
void main( )
{
    int count=0;
    int choose;
    while(1)
    {
        menu( );
        printf("请选择主菜单序号(0-6):");
        scanf("%d",&choose);
        switch(choose)
        {
            case 1:count=enter(s,N); break;
            case 2:list(s,count); break;
            case 3:search(s,count);break;
            case 4:count=del(count); break;
            case 5:save(count);break;
            case 6:read( ); break;
            case 0:return;
            default:printf("\n\n\n 输入无效请重新选择\n");
        }
```

```
        printf("\n\n 按任意键返回主菜单");
        getch();
    }
}
void menu()  //输入菜单
{
    system("cls");
    printf("\n\t\t ********************************MENU ******************************\n");
    printf("\n\t\t\t                1—添加歌曲 \n");                    //添加记录
    printf("\t\t\t                2—浏览歌曲\n");                      //显示记录
    printf("\t\t\t                3—查找歌曲 \n");                     //查找记录
    printf("\t\t\t                4—删除歌曲 \n");                     //删除记录
    printf("\t\t\t                5—保存文件 \n");                     //保存文件
    printf("\t\t\t                6—读取文件 \n");                     //读取文件
    printf("\t\t\t                0—退出\n");                          //退出
    printf("\n\t\t ************************************************************************\n");
}
/*-----------------添加歌曲-----------------------*/
int enter(struct song s[],int n)
{
    int i;
    printf("\n 请输入歌曲信息,按 Q 退出");
    for(i=0;i<n;i++)
    {
        printf("\n 请输入歌曲名称:");
        scanf("%s",s[i].name);
        if(s[i].name[0]=='Q'&&s[i].name[1]=='\0')
            break;
        printf("请输入作词:");
        scanf("%s",&s[i].writer);
        printf("请输入演唱者:");
        scanf("%s",&s[i].singer);
        printf("请输入发行年份:");
        scanf("%s",&s[i].year);
        printf("---------------------------------- \n");
    }
    return i;
}
/*-----------------浏览歌曲列表-----------------------*/
void list(struct song s[],int n)
{
    int i;
```

```
    printf("\n 歌曲信息浏览如下:");
    printf("\n\t 歌曲名\t\t 作词\t\t 演唱者\t\t 发行年份 \n");
    printf("-----------------------------------------------------------\n");
    for(i=0;i<n;i++)
    {
        printf("\t%s\t\t%s\t\t%s\t\t%s\n",s[i].name,s[i].writer,s[i].singer,s[i].year);
    }
    printf("-----------------------------------------------------------\n");
    printf("\n 浏览完毕,请按任意键返回主菜单");
    getch();
}
/*------------------查找歌曲信息----------------------*/
void search(struct song s[],int n)
{
    int find(int n,int b);              //声明分类查询 find()函数
    int m,i;
    printf("\n 选择查询代码:\n");
    printf("\n 1. 歌曲名");
    printf("\n 2. 作词者");
    printf("\n 3. 演唱者");
    printf("\n 0. 返回主菜单");
    do
    {
        printf("\n 请选择 0~3:\n");
        scanf("%d",&m);
    }while(m<0||m>3);
    switch(m)
    {
        case 1:printf("请输入歌曲名\n"); break;
        case 2:printf("请输入作词者"); break;
        case 3:printf("请输入演唱者");break;
        case 0:printf("返回主菜单");menu();
    }
    i=find(n,m);
    if(i>n-1)
        printf("没有查找到记录\n");
    else
    {
        printf("\n\t 歌曲名\t\t 作词\t\t 演唱者\t\t 发行年份 \n");
        printf("-----------------------------------------------------------\n");
        printf("\t%s\t\t%s\t\t%s\t\t%s\n",s[i].name,s[i].writer,s[i].singer,s[i].year);
        printf("-----------------------------------------------------------\n");
```

```
    }
    printf("\n ═══════════════════════查找完毕═══════════════════════");
    getch();
}
int find(int n,int b)     //分类查询函数
{
    int i;
    switch(b)
    {
        case 1:scanf("%s",s3.name);
            for(i=0;i<n;i++)
            if(strcmp(s3.name,s[i].name)==0)
                return i;
                    break;
        case 2:scanf("%s",s3.writer);
            for(i=0;i<n;i++)
                if(strcmp(s3.writer,s[i].writer)==0)
                    return i;
                break;
        case 3:scanf("%s",s3.singer);
            for(i=0;i<n;i++)
                if(strcmp(s3.singer,s[i].singer)==0)
                    return i;
                break;
    }
    return i;
}
/*------------------删除歌曲信息----------------------*/
int del(int n)
{
    int i,j,p,ch;
    printf("\n 选择删除代码:\n");
    printf("\n 1. 歌曲名");
    printf("\n 2. 作词者");
    printf("\n 3. 演唱者");
    printf("\n 0. 返回主菜单");
    do
    {
        printf("\n 请选择 0~3:\n");
        scanf("%d",&p);
    }while(p<0||p>3);
    switch(p)
```

```
    {
        case 1:printf("请输入歌曲名进行删除\n"); break;
        case 2:printf("请输入作词者进行删除"); break;
        case 3:printf("请输入演唱者进行删除");break;
        case 0:printf("返回主菜单");menu();
    }
    i=find(n,p);
    if(i>n-1)
        printf("没有找到要删除的记录\n");
    else
    {
        printf("\n\t 歌曲名\t\t 作词\t\t 演唱者\t\t 发行年份 \n");
        printf("-------------------------------------------------------\n");
        printf("\t%s\t\t%s\t\t%s\t\t%s\n",s[i].name,s[i].writer,s[i].singer,s[i].year);
        printf("-------------------------------------------------------\n");
        printf("确定要删除吗?(确定按1,取消按0)\n");
        scanf("%d",&ch);
        if(ch==1)
        {
            for(j=i+1;j<n;j++)
            {
                strcpy(s[j-1].name,s[j].name);
                strcpy(s[j-1].writer,s[j].writer);
                strcpy(s[j-1].singer,s[j].singer);
                strcpy(s[j-1].year,s[j].year);
            }
            n--;
        }
    }
    printf("已经删除\n");
    return n;
}
/*------------------保存到文件-----------------------*/
int save(int n)
{
    return 0;
}
/*------------------从文件读取-----------------------*/
int read()
{
    return 0;
}
```

7.1　结构体的基本概念

之前讨论的数据是单一的数据类型，而在实际应用中所涉及变量的属性是各种基本数据类型的组合，因而在 C 语言程序设计中引入了结构体类型的概念。

结构体类型是 C 语言的一种构造数据类型，它用于描述具有多个数据成员且每个数据成员具有不同数据类型的数据对象。例如，描写一个学生的基本情况，涉及学号、姓名、性别、两门课的成绩，分别用 int num；char name［8］；char sex；float score［2］；表示。要描写这样一个由不同数据类型构成的对象，需要定义一个结构体类型。

7.1.1　结构体类型的定义

结构体类型的定义格式如下：

```
struct   结构体类型名
{
    类型   数据类型成员名   1;
    类型   数据类型成员名   2;
    类型   数据类型成员名   3;
    ……
    类型   数据类型成员名   n;
};
```

例如，要描写学生的基本情况，需要定义的结构体类型如下：

```
struct   student{
    int num;
    char   name[8];
    char   sex;
    float   score[2];
};
```

注意：

此定义仅仅是结构体类型的定义，它说明了结构体类型的构成情况，C 语言并没有为之分配存储空间。

结构体中的每个数据成员称为“分量”或“域”，它们并不是变量，在实际应用中还需要定义结构体变量。

7.1.2　结构体变量的定义

在结构体类型定义完成后，就可以定义结构体变量。定义结构体变量的方法有两种，分别为结构体类型与结构体变量同时定义和分开定义。

1. 结构体类型与结构体变量同时定义

```
struct   student
{
    int      num;
    char   name[8];
```

```
    char   sex;
    float  score[2];
}stu;
```

2. 结构体类型与结构体变量分开定义

分开定义是指先定义结构体类型，再定义结构体变量。格式如下：

```
struct 结构体类型名  结构体变量表;
```

例如：

```
structstudent  stu;
```

其中，stu 是结构体类型 student 的实例或对象，称为结构体类型变量或结构体变量。

7.1.3 结构体变量占据的内存空间

定义了结构体变量（stu）之后，C 语言编译程序自动为结构体变量的所有成员分配足够的内存，如图 7-1-1 所示。结构体变量所占的存储空间是结构体类型各成员所占空间之和。

num	name	sex	score[0]	score[1]
4Byte	8Byte	1Byte	8Byte	8Byte

图 7-1-1 结构体变量 stu 占用内存的情况

在实际应用中，可用语句 sizeof（struct student）；测试结构体变量占用内存空间的大小。

7.1.4 结构体变量对结构体成员的引用

通过圆点（.）操作符可访问结构体中的成员。访问一个结构体成员的一般形式如下：

```
结构体变量名.成员名;
```

它表示结构体变量对具体成员的引用，以下代码表示把 2001 赋值给结构体变量 stu1 的成员 sum；

```
stu1.num=2001;
```

在屏幕上显示 stu 的成员 num 所含的学号值，应写为：

```
printf("%d",stu.num);
```

同理，从键盘读入的语句是：

```
scanf("%d",&stu.num);
```

7.1.5 结构变量的赋值

给结构体变量的赋值方法有三种：初始化，逐个赋值，从键盘读入。

（1）结构体变量可在声明时直接进行初始化。初始化数据应放在大括号中，并根据成员变量的声明次序排列，同时数据之间类型应一致。例如：

```
structstudent  stu1={2001,"张华",'M',86.00,92.2};
```

或

```
structstudent
{
    int   num;
```

```
    char name[8];
    char sex;
    float  score[2];
}stu1={2001,"张华",'M',86.00,92.2};
```

【例 7.1】定义结构体类型并定义它的变量，在声明时直接进行初始化，最后在终端输出结构体变量各成员的值。

分析：初始化结构体变量 stu1，在初始化时直接给各成员变量赋初值，在主程序中用 printf（）直接输出结构体各成员变量的值。

源程序代码：

```
#include<stdio.h>
struct student
{
    int num;
    char name[8];
    char sex;
    float score[2];
}stu1={2001,"张华",'M',86.00,92.2};
int main()
{
    printf("%d\t%s\t%c\t%f\t%f\n",stu1.num,stu1.name,stu1.sex,stu1.score[0],stu1.score[1]);
    return 0;
}
```

程序编译成功后运行，程序的输出结果如图 7-1-2 所示。

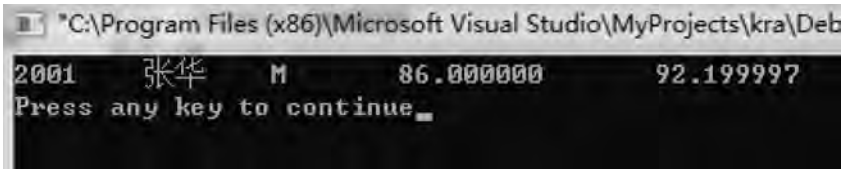


图 7-1-2　例 7.1 运行结果

（2）对结构体变量的数据成员进行逐个赋值。

【例 7.2】定义一个结构体类型及结构体变量，逐个给结构体变量赋值，最后输出结构体变量的值。

分析：定义结构体类型 student 及结构体变量 stu1，在程序中逐个给成员变量赋值，最后用 printf（）直接输出结构体各成员变量的值。

源程序代码：

```
#include<stdio.h>
#include<string.h>
struct student
{
    int num;
```

```
    char name[8];
    char sex;
    float score[2];
}stu1;
int main()
{
    stu1.num=2001;
    strcpy(stu1.name,"张华");
    stu1.sex='M';
    stu1.score[0]=86.00;
    stu1.score[1]=92.2;
    printf("%d\t%s\t%c\t%f\t%f\n",stu1.num,stu1.name,stu1.sex,stu1.score[0],stu1.score[1]);
    return 0;
}
```

程序编译成功后运行，程序的输出结果如图 7-1-3 所示。

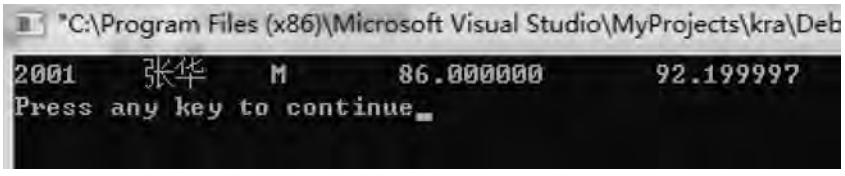


图 7-1-3　例 7.2 运行结果

（3）可用单赋值语句把一个结构体变量的全部内容赋值给另一个同类结构体变量，而不必逐个成员地多次赋值。

【例 7.3】利用结构体变量存储，并在初始化时给成员变量赋值，把所有成员变量的值传递给另一个结构体变量，利用第二个结构体变量，在终端输出结构体各成员变量的值。

分析：初始化结构体变量 stu1，在初始化时直接给各成员变量赋初始值，在主程序中定义一个结构相同的结构体变量 stu2，用单赋值语句把结构体变量 stu1 的全部内容赋值给第二个同类结构体变量 stu2，最后用 printf（）直接输出结构体变量 stu2 各成员变量的值。

源程序代码：

```
#include<stdio.h>
struct student
{
    int num;
    char name[8];
    char sex;
    float score[2];
}stu1={2001,"张华",'M',86.00,92.2};
int main()
{
    struct student stu2;
```

```
    stu2=stu1;
    printf("%d\t%s\t%c\t%f\t%f\n",stu2.num,stu2.name,stu2.sex,stu2.score[0],stu2.score[1]);
    return 0;
}
```

程序编译成功后运行，程序的输出结果如图 7-1-4 所示。

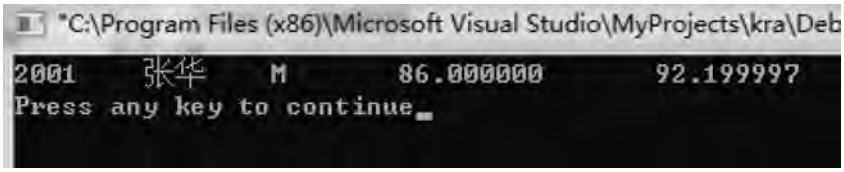


图 7-1-4　例 7.3 运行结果

（4）从键盘中逐个读入结构体数据成员。

【例 7.4】利用结构体变量存储，从键盘中逐个读入结构体数据成员，给成员变量赋值，在终端输出结构体各成员变量的值。

分析：初始化结构体变量 stu1，在初始化时并不赋初值，在主程序中等待键盘输入，依次把键盘的输入赋值给结构体变量 stu1 的各成员变量，最后用 printf（）直接输出结构体变量 stu1 的各成员变量的值。

源程序代码：

```
#include<stdio.h>
struct student
{
    int num;
    char name[8];
    char sex;
    float score[2];
}stu1;
int main()
{
    char  ch;
     scanf("%d%s%c%c%f%f",&stu1.num,stu1.name,&ch,&stu1.sex,&ch,stu1.score[0],
&stu1.score[1]);
    printf("%d\t%s\t%c\t%f\t%f\n",stu1.num,stu1.name,stu1.sex,stu1.score[0],stu1.score[1]);
    return 0;
}
```

7.2　结构体变量的引用

实际开发的很多结构化程序，会用到不少自定义的函数来实现特定的功能。因此，很多时候，结构体变量在程序中也可以作为函数的参数使用。

要将一个结构体变量的值传递给另一个函数，有以下三种方法：

（1）用结构体变量的成员作参数，将实参值传给形参，这种用法和用普通变量作实参一样，属于传值方式。

（2）用结构体变量作参数，其前提是函数的形参和调用函数的实参必须是同类型的结构体变量，这也是一种传值方式，将实参结构体变量所占内存单元的内容全部顺序地传给形参。

（3）用指向结构体变量（或数组）的指针作实参，将结构体变量（或数组）的地址传给形参，属于传址方式。

【例 7.5】 定义结构体变量 Student，包括年龄、姓名和性别。在 InputStudent 函数中输入信息，并在 OutputStudent 函数中输出信息。

```
#include <stdio.h>
#include <string.h>
//定义结构体类型 struct Student
struct Student
{
    int age;
    char name[100];
    char sex;
};
//声明输入/输出函数
void InputStudent(struct Student * pst);
void OutputStudent(struct Student st);
//main 函数,程序入口
int main(void)
{
    struct Student st;
    //想要修改学生结构体变量 st 的值,
    //必须通过传递 st 地址,去修改
    InputStudent(&st);
    //输出学生信息既可以传递结构体变量
    //也可以传递结构体变量地址
    OutputStudent(st);
    return 0;
}
//输出学生结构体成员信息的函数
void OutputStudent(struct Student st)
{
    printf("%d %s %c\n",st.age,st.name,st.sex);
}
//输入学生结构体成员信息的函数
void InputStudent(struct Student * pst)
{
```

```
    pst->age = 10;
    strcpy(pst->name,"张三");
    pst->sex = 'M';
}
```

分析：虽然说输出学生信息的函数可以写成接收结构体变量的函数，但是由于使用 sizeof（struct Student）或者 sizeof（st）占用的内存空间为 108 个字节，void Output Student（struct Student * pst）比 sizeof（st）占用的内存空间小，所以写成指针变量更好些。另外这样写有个风险就是 pst 可以修改结构体，如果有人在输出函数中修改了结构体则很不安全，C 语言中 const 常变量修饰符可以防止被修改。

7.3　结构体数组

结构体的最常见用法就是结构体数组（array of structures）。例如，描述一个班级的学生。用结构体类型中不同类型的成员变量描述学生的具体属性，用数组类型描述拥有相同属性的一个班级的学生情况，以便使用循环对数组元素进行统一处理，优化算法。

定义结构体数组时，可写成：

```
struct  student  stu[40];
```

或

```
struct  student
{
    int  num;
    char  name[8];
    char  sex;
    float  score[2];
}stu[40];
```

结构体数组的初始化与其他类型的数组类似。例如：

```
struct  student
{
    int  num;
    char  name[8];
    char  sex;
    float  score[2];
}stu[2]={{4001,"刘','M',86.5,97.3},{4002,"张","F',78.4,86.5}};
```

【例 7.6】用结构体数组编写候选人得票统计程序。设有 3 个候选人，有 10 个选民，每个选民只能选一个候选人，不考虑弃权情况，要求最后输出每个候选人的得票结果。

分析：在程序中定义一个全局结构体类型 person，它有两个成员 name（姓名）和 count（得票数）。结构体数组在 main（）函数中定义并初始化。字符数组 l_name 代表每次输入的被选人的姓名，在每次输入后，与候选人的名字相比，相同则相应候选人的得票数加 1。输入统计结束后，输出 3 个候选人及相应的得票数。

源程序代码：

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

```
#include<string.h>
struct person
{
    char name[20];
    int count;
};
int main()
{
    int k,t;
    char lname[20];
    struct  person  leader[3]={"A",0,"B",0,"C",0};
    printf("请给A、B和C三个人投票:\n");
    for(k=1;k<=10;k++)
    {
        scanf("%s",lname);
        for(t=0;t<3;t++)
        {
            if(strcmp(lname,leader[t].name)==0)
            leader[t].count++;
        }
    }
    printf("\n投票结果是:\n");
    for(t=0;t<3;t++)
    printf("%10s:%5d票",leader[t].name,leader[t].count);
    printf("\n");
    return 0;
}
```

程序编译成功并运行，结果如下：

```
A
B
C
A
A
投票结果是:
A: 3票          B: 1票          C: 1票
```

7.4 共用体

共用体（union）是一种特殊的数据类型，允许在相同的内存位置存储不同的数据类型。可以定义一个带有多成员的共用体，但是任何时候只能有一个成员带有值。共用体提供了一种使用相同的内存位置的有效方式。

7.4.1　共用体的概念

有时需要将几种不同类型的变量存放到同一段内存单元中。例如，可把一个整型变量、一个字符型变量、一个双精度型变量放在同一个地址开始的内存单元，如图 7-4-1 所示。

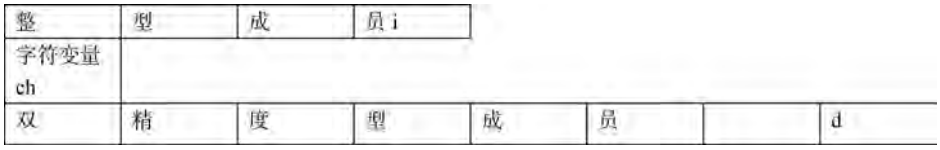


图 7-4-1　共用体内存

以上 3 个变量在内存中占用的字节数不同，但都从同一地址开始（图中设地址为 1000）存放，也就是使用覆盖技术，几个变量互相覆盖。这种使几个不同的变量共占同一段内存的结构，称为共用体类型的结构（有些书译为联合，笔者认为称为共彡体更能体现出其特征，也易于理解）。

声明共用体类型的一般形式为：

```
union　共用体类型名
{
    成员表列
};
```

定义共用体变量的一般形式为：

```
共用体类型名　共用体变量名;
```

当然也可以在声明共用体类型的同时定义共用体变量，也可以没有共用体类型名而直接定义共用体变量。例如：

```
union　data
{
    int　i;
    char ch;
    double　d;
}a,b,c;
(有共用体类型名)
```

```
union
{
    int　i;
    char　ch;
    double　d;
}a,b,c;
(无共用体类型名)
```

可以看到，“共用体”与“结构体”的定义形式相似，但它们的含义是不同的。结构体变量所占内存长度是各成员的内存长度之和。每个成员分别占有其自己的内存单元。共用体变量所占的内存长度等于最长的成员的长度。例如，上面定义的“共用体”变量 a、b、c 各占 8 字节（因为一个双精度型变量占 8 个字节），而不是占 4+1+8=13 个字节。

7.4.2　对共用体变量的访问方式

不能引用共用体变量，而只能引用共用体变量中的成员。例如，下面的引用方式是正确的：

```
a.i　//引用共用体变量中的整型成员 i
a.ch　//引用共用体变量中的字符型成员 ch
a.d　//引用共用体变量中的双精度型成员 d
```

不能只引用共用体变量，例如：

printf（a）；

是错误的，应该写成 printf（a. i）；或 printf（a. ch）；等。

7.4.3 共用体的应用

共用体在一般的编程中应用较少，在单片机中应用较多。对于 PC 机，经常使用到的一个实例是，现有一张关于学生信息和教师信息的表格，学生信息包括姓名、编号、性别、职业、分数，教师的信息包括姓名、编号、性别、职业、教学科目，见表 7-4-1。

表 7-4-1 学生/教师信息表

name	num	sex	professio	score / course
HanXiaoXiao	501	f	s	89. 5
YanWeiMin	1011	m	t	math
LiuzHenTao	109	f	t	english
ZhaoFeiYan	982	m	s	95. 0

f 和 m 分别表示女性和男性，s 表示学生，t 表示教师。可以看出，学生和教师所包含的数据是不同的。现在要求把这些信息放在同一张表格中，并设计程序输入人员信息然后输出。

如果把每个人的信息都看作一个结构体变量的话，那么教师和学生的前 4 个成员变量是一样的，第 5 个成员变量可能是 score 或者 course。当第 4 个成员变量的值是 s 的时候，第 5 个成员变量就是 score；当第 4 个成员变量的值是 t 的时候，第 5 个成员变量就是 course。

经过上面的分析，我们可以设计一个包含共用体的结构体。

【例 7.7】设有若干个人员的数据，包含学生和教师。学生信息包括姓名、编号、性别、职业、分数，教师的信息包括姓名、编号、性别、职业、教学科目。

```
#include <stdio. h>
#include <stdlib. h>
#define TOTAL 4   //人员总数
struct{
    char name[20];
    int num;
    char sex;
    char profession;
    union{
        float score;
        char course[20];
    } sc;
} bodys[TOTAL];
int main(){
```

```
    int i;
    //输入人员信息
    for(i=0; i<TOTAL; i++){
        printf("Input info:");
        scanf("%s %d %c %c",bodys[i].name,&(bodys[i].num),&(bodys[i].sex),&(bodys[i]
.profession));
        if(bodys[i].profession == 's'){   //如果是学生
            scanf("%f",&bodys[i].sc.score);
        }else{   //如果是老师
            scanf("%s",bodys[i].sc.course);
        }
        fflush(stdin);
    }
    //输出人员信息
    printf("\nName\t\tNum\tSex\tProfession\tScore / Course\n");
    for(i=0; i<TOTAL; i++){
        if(bodys[i].profession == 's'){   //如果是学生
            printf("%s\t%d\t%c\t%c\t\t%f\n",bodys[i].name,bodys[i].num,bodys[i].sex,bodys
[i].profession,bodys[i].sc.score);
        }else{   //如果是老师
            printf("%s\t%d\t%c\t%c\t\t%s\n",bodys[i].name,bodys[i].num,bodys[i].sex,bodys
[i].profession,bodys[i].sc.course);
        }
    }
    return 0;
}
```

运行结果：

```
Input info:HanXiaoXiao 501 f s 89.5↙
Input info:YanWeiMin 1011 m t math↙
Input info:LiuZhenTao 109 f t English↙
Input info:ZhaoFeiYan 982 m s 95.0↙
```

name	num	sex	profession	score / course
HanXiaoXiao	501	f	s	89.500000
YanWeiMin	1011	m	t	math
LiuZhenTao	109	f	t	English
ZhaoFeiYan	982	m	s	95.000000

7.5 枚举类型

如果一个变量只有几种可能的值，可以定义为枚举类型。所谓“枚举”，是指将变量可能取的值一一列举出来，使用时变量的值只能取列举出来的值。

枚举定义的一般形式为：

```
enum 枚举类型名{取值表 };
```

enum 是定义枚举类型的关键词。花括号中的内容称为枚举表，每个枚举表项是常整数，以逗号分隔，系统规定其值依次为 0，1，2，3，…。

枚举在日常生活中的应用十分常见，如一周分为 7 天：sun，mon，tue，wed，thu，fri，sat。

可按如下格式定义为枚举类型：

```
enum day{sun,mon,tue,wed,thu,fri,sat};
```

系统规定其值依次为 sun=0，mon=1，tue=2，wed=3，thu=4，fri=5，sat=6。

说明：

（1）在定义枚举类型时可对枚举表项进行初始化以改变它们的值。例如：

```
enum day{sun=1,mon,tue=4,wed,thu,fri,sat=9};
```

则 sun=1，mon=2，tue=4，wed=5，thu=6，fri=7，sat=9；

（2）枚举元素都是常量，即枚举常量。因为不是变量，所以不能为枚举元素赋值，如 sun=5，mon=8 是错误的。

枚举元素可用于给枚举变量赋值，而枚举变量只能接受一个枚举常量的赋值。如：

```
today=sun;(正确)
today=2;(错误)
```

（3）可以将一个整数经强制类型转换后赋给枚举变量。如：

```
enum day{sun,mon,…}today;  /* today 为枚举变量 */
today=(enum day)2 相当于:today=tue;
```

（4）与结构体类型一样，枚举也必须先定义类型，再定义变量。如：

```
enum day{sun,mon,tue,wed,thu,fri,sat};
enum day today;
```

【例 7.8】 从键盘上输入一整数，显示与该整数对应的一周内枚举常量的英文名称。

```
#include<stdio.h>
main()
{
    enum week{sun,mon,tue,wed,thu,fri,sat};     /*定义枚举类型 week*/
    enum week weekday;          /*定义 week 类型变量 weekday*/
    int i;
    scanf("%d",&i);
    printf("\nYour input is:");
    weekday=(enum week)i;      /*将整数 i 强制转换为 week 类型*/
    switch(weekday)
    {
        case sun:printf("Sunday");                  break;
        case mon:printf("Monday");break;
        case tue:printf("Tuesday");                 break;
```

```
            case wed:printf("Wednesday");                    break;
            case thu:printf("Thursday");break;
            case fri:printf("Friday");break;
            case sat:printf("Saturday");break;
            default:printf("\nInput error!");
        }
    }
```

运行结果：

```
2↙
Your input is:Tuesday
```

C 语言提供了许多标准类型名，如 int、char、float 等，用户可以直接使用这些类型名定义所需要的变量。同时 C 语言还允许使用 typedef 语句定义新类型名，以取代已有的类型名，如：

```
Typedef  int COUNTER;
```

作用是使 COUNTER 等价于基本数据类型名 int，以后就可以利用 COUNTER 定义变量了。如：

```
COUNTER i,n;  等价于 int i,n;
```

使用类型定义的优点是能够提高程序的可读性。由上述语句可以看出，当用 COUNTER 来定义 i、n 变量时，就可以判断出 i、n 变量的作用是计数器，但如果用 int 来定义，就难以看出这种用途。

说明：

（1）typedef 语句不能创造新的类型，只能为已有的类型增加一个类型名。

（2）typedef 语句只能用来定义类型名，而不能用来定义变量。

（3）利用 typedef 可以简化结构体变量的定义。如有如下结构体：

```
struct employee
{
    int num;
    char name[10]; char sex; int age;
};
```

如果要定义结构体变量 emp1，emp2，应采用：

```
struct employee emp1,emp2;
```

这样做需要键入的内容较多。这时，可以使用 typedef 来简化变量的定义，在定义结构体类型时直接定义为新类型，如以下语句：

```
typedef  struct  employee
{
    int num;
    char name[10]; char sex; int age;
} EMP;
EMP emp1,emp2;          /*即用 EMP 来代替类型 struct employee */
```

这种形式可以达到一样的效果，且更简洁。在程序的说明中，我们定义的新类型名用大写表示，这并不是系统的要求，目的是为了与其他的变量相区别。

任务八　歌曲数据的存储

学习目标

- 理解和掌握文件的打开、关闭；
- 掌握文件的读写操作。

一、任务描述

本项目也是通过函数来实现模块化程序设计的。通过自定义函数 enter（）添加歌曲信息，添加完成后将歌曲信息列表保存在磁盘文件中。在浏览、删除时，首先将数据读入结构体数组中，对结构体数组进行操作后，再将其中的数据保存到文件中。该任务用 save（）函数将结构体数组存入 songlist. txt 文件中，用 read（）函数将 songlist. txt 文件的数据导入结构体数组中。

二、知识要点

本任务主要涉及文件的打开、读写、关闭等知识。

三、任务分析

实现歌曲信息的处理和保存时，分别用 save（）和 read（）函数实现存储和读取。由于从文件中读取或者通过函数 insert（）输入的学生数量是不定的，所以在 insert（）、save（）函数中均要统计输入或读取的歌曲数量。

四、源代码参考

```
int save(int n)
{
    int i;
    FILE * fp;
    if((fp=fopen("songlist.txt","wt"))==NULL)
    {
        printf("无法打开文件");
        return 0;
    }
    printf("\n 保存文件\n");
    for(i=0;i<n;i++)
    fprintf(fp,"%s %s %s %s\n",s[i].name,s[i].writer,s[i].singer,s[i].year);
    fclose(fp);
```

```
    printf("═══════════════════════════* 保存成功 *═══════════════════════════");
    //menu();
    return 1;
}
/*------------------从文件读取-----------------------*/
int read()
{
    int i;
    FILE *fp;
    if((fp=fopen("songlist.txt","rt"))==NULL)
    {
        printf("无法打开文件");
        return 0;
    }
    printf("\n\t 歌曲名\t\t 作词\t\t 演唱者\t\t 发行年份 \n");
    printf("-----------------------------------------------------------------\n");
    for(i=0;! feof(fp);i++)
    {
        fscanf(fp,"%s %s %s %s\n",s[i].name,s[i].writer,s[i].singer,s[i].year);
        printf("%s %s %s %s\n",s[i].name,s[i].writer,s[i].singer,s[i].year);
    }
    printf("-----------------------------------------------------------------\n");
    fclose(fp);
    printf("═══════════════════════════* 读取完毕 *═══════════════════════════");
    return 1;
}
```

8.1　文件的概念

文件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。如果想找存在外部介质上的数据，必须先按文件名找到所指文件，然后再从文件中读取数据。要向外部介质上存储数据也必须先建立一个文件，才能向它输出数据，如图 8-1-1 所示。

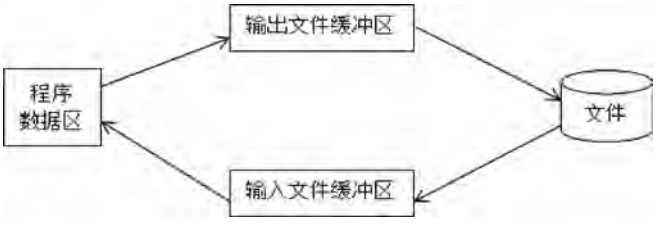


图 8-1-1　文件存储示意图

1. 文件的分类

从用户观点文件可分为特殊文件（标准输入/输出文件或标准设备文件）和普通文件（磁盘文件）。

从操作系统的角度看，每一个与主机相连的输入/输出设备都可看作是一个文件。

例：输入文件：终端键盘。

输出文件：显示屏和打印机。

（1）根据文件的内容，文件可分为程序文件和数据文件，程序文件又可分为源文件、目标文件和可执行文件。

（2）根据文件的组织形式，文件可分为顺序存取文件和随机存取文件。

（3）根据文件的存储形式，文件可分为 ASCII 码文件和二进制文件。

ASCII 码文件的每 1 个字节存储 1 个字符，因而便于对字符进行逐个处理，但一般占用存储空间较多，而且要花费转换时间（二进制与 ASCII 码之间的转换）。二进制文件是把内存中的数据原样输出到磁盘文件中。这样可以节省存储空间和转换时间，但 1 个字节并不对应 1 个字符，不能直接输出字符形式。

2. C 语言对文件的处理方法：

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

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

8.2 文件的打开和关闭

C 语言规定，对磁盘文件进行读写之前首先应该“打开”该文件，然后再进行具体的“读/写”操作；在文件使用结束后，应该“关闭”该文件。

8.2.1 文件类型指针

在 C 语言中，对文件操作必须定义一个文件指针变量，只有通过文件指针变量，才能实现对文件的访问。

C 语言的文件管理系统为每个文件在内存中开辟一个区，用来存放诸如文件的名字、文件的状态及文件当前位置等有关信息。这些信息被保存在一个由系统定义的、取名为 FILE 的结构体类型的变量中。FILE 定义形式如下：

```
typedef struct
{
    short level;                /*缓冲区“满”或“空”的程度*/
    unsigned flags;             /*文件状态标志*/
    char fd;                    /*文件描述符*/
    unsigned char hold;         /*如无缓冲区不读取字符*/
    short bsize;                /*缓冲区的大小*/
    unsigned char  *buffer;     /*数据缓冲区的位置*/
    unsigned ar  *curp;         /*指针当前的指向*/
```

```
    unsigned istemp;            /* 临时文件,指示器 */
    short token;                /* 用于有效检查 */
}FILE;
```

例如，我们可以定义一个 FILE 类型的数组：

```
FILE fi[3];
```

该数组定义了一个结构体数组 fi，它有 3 个元素，可以用来存放 3 个文件的信息。

又例如，可以定义一个文件型指针变量：

```
FILE *fp;
```

fp 是指向 FILE 类型结构体的指针变量，可以使 fp 指向某一文件的结构体变量，从而通过结构体变量中的文件信息能够访问该文件。

8.2.2　文件的打开

C 语言在标准输入/输出函数库中定义了对文件操作的若干函数，其中 fopen（）函数用来打开磁盘文件。一般格式：

```
FILE  *fp;
fp=fopen("文件名","文件使用方式");
```

功能：以指定的文件使用方式打开一个文件。

说明：

（1）fp 是 FILE 文件类型指针，用来指向被打开文件数据区（结构变量）的起始地址。

（2）“文件名”为要打开文件的文件名，若不在当前默认路径，则要把路径书写完整。

（3）“文件使用方式”指文件类型和操作方式，见表 8-2-1。

表 8-2-1　文本文件的使用方式

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）	为输出打开或建立一个文本文件
a（追加）	向一个文本文件尾部追加数据
rb（只读）	为输入打开一个二进制文件
wb（只写）	为输出打开或建立一个二进制文件
ab（追加）	向一个二进制文件尾部追加数据
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/写打开或建立一个新的文本文件
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/写打开或建立一个二进制文件

例如：

```
FILE *fp;
fp=fopen("b1","r");
```

表示要打开名字为 b1 的文件，文件使用方式为“只读”。fopen（）函数返回一个指向 b1 文件的指针赋给 fp，这样 fp 就指向 b1 文件。

当打开一个文件时，可以通过 fopen 函数是否返回一个 NULL 空指针值来判断文件是否被正常打开。例如：

```
if(fp=fopen("b1","r")==NULL)  /*判断文件名为 b1 的文件是否被正常打开*/
{
    printf("The file cannot be opened");
    exit(0);
}
```

8.2.3 文件的关闭

使用完一个文件后，应该及时关闭，以防止再被误用，导致数据丢失。一般格式：

```
fclose(文件指针);
```

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

返回值：关闭成功返回值为 0，否则返回非 0 值。

8.3 文件的顺序读写

文件成功打开后，就可以对它进行读写操作了，文件的顺序读写指的是按数据流的先后顺序对文件进行读写操作。在 C 语言中，对文件的读写操作是通过函数调用实现的。

8.3.1 fputs（）函数和 fgets（）函数

1. fputs 函数

一般格式：fputs（str，fp）；

功能：把一个字符串写到指定的磁盘文件中。

说明：

（1）str 为字符数组或字符型指针，fp 为 FILE 类型的文件指针变量。

（2）fputs（）把某一个字符串输出到指定的文件中。

（3）fputs（）函数带有返回值。若输出成功，返回值为 0，否则为非零值。

2. fgets（）函数

一般格式：fgets（str，m，fp）；

功能：从指定的磁盘文件中读取一个字符串。

说明：

（1）str 为字符数组或字符型指针。

（2）fp 为 FILE 类型的文件指针变量。

（3）m 为正整数，表示从文件中读取不超过 m-1 个字符，在读取的最后一个字符后加上串结束标志‘\0’。如果在完成读取 m-1 个字符之前，遇到换行符或 EOF，则读入

过程立即结束。fgets（）的返回值为 str 的首地址。若只读到文件尾或出错，则返回空指针 NULL。

【例 8.1】从键盘上输入三行字符，并存入指定的文件 file.doc 中。

```
#include <stdlib.h>
#include <stdio.h>
main()
{
    int i;
    char str[81];
    FILE *fp;
    if((fp=fopen("file.doc","w"))==NULL)      /*创建 doc 文件且判断能否正常打开*/
    {
        printf("The file cannot be opened");
        exit(0);
    }
    for(i=1;i<4;i++)                          /*循环 3 次,写入 3 行字符串*/
    {
        gets(str);                            /*接收字符串保存在数组中*/
        fputs(str,fp);                        /*把字符串写到文件上*/
        fputs("\n",fp);
    }
    fclose(fp);                               /*关闭文件*/
}
```

运行后输入：

```
I love C! ↙
I will study hard.↙
Thank you.↙
```

运行结束后，会见到 file.doc 文件已被创建，并且打开该文档，能看到以下文字：

```
I love C!
I will study hard.
Thank you.
```

这说明，字符串被成功写到 file.doc 文件里。

【例 8.2】续例 8.1，文件 file.doc 已经存在并存有三行字符，现要从文件 file.doc 中读取一串字符，并显示在屏幕上。

```
#include<stdio.h>
main()
{
    char str[30];
    FILE *fp;
    if((fp=fopen("file.doc","r"))==NULL)      /*创建 doc 文件且判断能否正常打开*/
    {
        printf("The file cannot be opened\n");
```

```
        exit(0);
    }
    while(fgets(str,30,fp)! =NULL)          /*读取字符串*/
    printf("%s",str);                       /*输出已读取的字符串*/
    fclose(fp);                             /*关闭文件*/
}
```

运行结果：

```
I love C!
I will study hard.
Thank you.
```

这表明，file. doc 文件里的字符串被成功读取，且输出显示在屏幕上。

8.3.2 fwrite（）函数和 fread（）函数

在编程时经常需要读写由各种类型数据组成的字段，此时可以用 fread（）和 fwrite（）两个函数来实现数据字段的读写。

1. fwrite（）函数

一般格式：fwrite（buffer，size，count，fp）；

功能：将一组数据输出到指定的磁盘文件中。

说明：

（1）buffer 用于存放输出数据的缓冲区指针，指向输出数据的起始地址。

（2）size 是输出的每个数据项的字节数。

（3）count 是指要输出多少个 size 字节的数据项。

（4）fp 是 FILE 类型的文件指针变量。

2. fread（）函数

一般格式：fread（buffer，size，count，fp）；

功能：从指定的文件中读入一组数据。

说明：

（1）buffer 用于存放读入数据的缓冲区指针，指向读入数据的起始地址。

（2）size 是读入的每个数据项的字节数。

（3）count 是指要读多少个 size 字节长的字段。

（4）fp 是 FILE 类型的文件指针变量。

【例 8.3】 从键盘输入 8 个整数存入文件 file. dat 中，然后再从该文件中读取后 6 个数逆序输出。

```
#include "stdio. h"
main( )
{
    FILE *fp;
    int d[8],I;
    for(i=0;i<8;i++)
    scanf("%d",&d[i]);                      /*存入 8 个整数到数组 d 中*/
    if((fp=fopen("file. dat","w+"))= =NULL)  /*判断是否能正常打开文件*/
```

```
        printf("The file cannot be opened.");
        else
        {
            fwrite(d,4,8,fp);                    /*把8个整数写入指定的文件中*/
            fclose(fp);
        }
        if((fp=fopen("file.dat","w+"))==NULL)
        printf("The file cannot be opened.");
        else
        {
            fread(d,4,8,fp);                     /*从文件中读入数据到数组d中*/
            for(i=5;i>=0;i--)                    /*逆序输出后6个整数*/
            printf("%-3d",d[i]);
        }
}
```

8.3.3　fprintf（）函数和 fscanf（）函数

对文件进行格式化输入输出时，要用到 fprintf（）函数和 fscanf（）函数。从函数名可以看出，它们只是在 printf 和 scanf 的前面加了一个字母 f。它们的作用与 printf（）函数和 scanf（）函数相仿，都是格式化读写函数。只有一点不同：fprintf（）和 fscanf（）函数的读写对象不是终端而是文件。

1. fprintf（）函数

一般格式：fprintf（文件类型指针，格式控制，输出表列）；

功能：将“输出表列”变量中的数据，输出到“文件类型指针”所标识的文件中。

例如：将 int 型变量 i 和 float 型变量 f 的值按%d 和%6.2f 的格式输出到 fp 指向的文件中。

```
fprintf(fp,"%d,%6.2f",I,f);
```

一般来讲，由 fprintf（）函数写入磁盘文件中的数据，应由 fscanf（）函数以相同格式从磁盘文件读出来使用。

2. fscanf（函数）

一般格式：fscanf（文件类型指针，格式控制，地址表列）；

功能：从“文件类型指针”所标识的文件读入一个字符流，存入“地址表列”对应变量中。

例如：磁盘文件上如果有字符“3，4.5”，则从中读取整数 3 送给整型变量 i，读取实数 4.5 送给 float 型变量 f。

```
fscanf(fp,"%d,%f",&i,&f);
```

注意：在利用 fscanf（）函数从文件中进行格式化输入时，一定要保证格式说明符与所对应输入数据的一致性，否则会出错。通常的做法是用什么格式写入的数据，就用什么格式来读出。

8.4　文件的定位及随机读写

对文件进行顺序读写比较容易理解，也容易操作，但有时效率不高。而随机访问不是

按数据在文件中的物理位置次序进行读写，而是可以对任何位置上的数据进行访问，显然这种方法比顺序访问效率高得多。

1. 文件的定位

为了对读写进行控制，系统为每个文件设置了一个文件读写位置标记，用来指示接下来要读写的下一个字符的位置。一般情况下，在对字符文件进行顺序读写时，文件位置标记指向文件开头，这时如果对文件进行读/写的操作，读/写完第 1 个字符后，文件位置标记顺序向后移一个位置，在下一次执行读/写操作时，就将位置标记指向的第 2 个字符进行读出或写入。依此类推，直到文件尾，此时文件位置标记在最后一个数据之后，如图 8-4-1 所示。

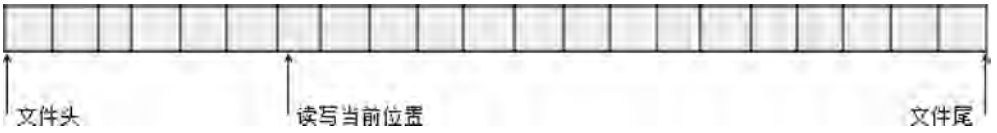


图 8-4-1　文件位置标记示意图

对流式文件既可以进行顺序读写，也可以进行随机读写。关键在于控制文件的位置标记。如果文件位置标记是按字节位置顺序移动的，就是顺序读写。如果能将文件位置标记按需要移动到任意位置，就可以实现随机读写。所谓随机读写，是指读写完上一个字符（字节）后，并不一定要读写其后续的字符（字节），而可以读写文件中任意位置上所需要的字符（字节）。即对文件读写数据的顺序和数据在文件中的物理顺序一般是不一致的，可以在任何位置写入数据，在任何位置读取数据。

（1）用 rewind（）函数使文件位置标记指向文件开头。

一般形式：rewind（文件指针）；

说明：rewind（）函数的作用是使文件位置标记重新返回文件的开头，此函数没有返回值。

（2）用 fseek（）函数改变文件位置标记。

一般形式：fseek（文件类型指针，位移量，起始点）；

说明：“起始点”用 0，1 或 2 代替，0 代表文件开始位置，1 为当前位置，2 为文件末尾位置。“位移量”指以“起始点”为基点，向前移动的字节数（长整型）。

fseek（）函数一般用于二进制文件。例如：

```
fseek(fp,100L,0);          /*将文件位置标记向前移到离文件开头 100 个字节处*/
fseek(fp,50L,1);           /*将文件位置标记向前移到离当前位置 50 个字节处*/
fseek(fp,-10L,2);          /*将文件位置标记从文件末尾处向后退 10 个字节*/
```

（3）用 ftell（）函数测定文件位置标记的当前位置。

ftell（）函数的作用是得到流式文件中文件位置标记的当前位置，用相对于文件开头的位移量来表示。如果调用函数时出错（如不存在 fp 指向的文件），ftell（）函数返回值为-1L。

例如：

```
i=ftell(fp);          /*变量 i 存放文件当前位置*/
if(i==-1L)printf("error\n");/*如果调用函数时出错,输出"error"*/
```

【例 8.4】在磁盘文件上存有 10 个学生的数据。要求将第 1、3、5、7、9 个学生数据输入计算机，并在屏幕上显示出来。

```
#include<stdio.h>
#include <stdlib.h>
struct Studenttype  /*学生数据类型*/
{char name[10];
    int num;
    int age;
    char addr[15];
}stud[10];
int main()
{int i;
    FILE *fp;
    if((fp=fopen("stu.dat","rb"))==NULL)/*以只读方式打开二进制文件*/
    {printf("can not open file\n");
        exit(0);
    }
    for(i=0;i<10;i+=2)
    {
        fseek(fp,i*sizeof(struct Studenttype),0);/*移动文件位置标记*/
        fread(&stud[i],sizeof(struct Studenttype),1,fp);/*读一个数据块到结构体变量*/
        printf("%-10s %4d %4d %-15s\n",stud[i].name,stud[i].num,stud[i].age,stud[i].addr);
                                                    /*在屏幕输出*/
    }
    fclose(fp);
    return 0;
}
```

2. 文件的随机读写

利用 fseek（）函数，可以实现文件的随机读写，fseek（）函数可以按偏移量来移动文件的位置指针，函数调用的一般形式为：

```
fseek(文件指针,位移量,起始点);
```

其中，“文件指针”指向被移动的文件，“位移量”表示移动的字节数，是 long 型数据，以便在文件长度大于 64 KB 时不会出错。当用常量表示位移量时，要求加后缀“L”。“起始点”表示从何处开始计算位移量，规定的起始点有三种：文件首、当前位置和文件尾。文件位置的详细表示方法见表 8-2-2。

表 8-2-2　文件位置的表示方法

起始点	名字	用数字代表
文件开始位置	SEEK_ SET	0
文件当前位置	SEEK_ CUR	1
文件末尾位置	SEEK_ END	2

fseek（）函数一般用于二进制文件。由于在文本文件中要进行转换，所以计算的位置可能会出现错误。

移动位置指针后，即可用前面介绍的任一种读写函数进行读写。由于一般是读写一个数据块，所以常用 fread（）和 fwrite（）函数来操作。

附　　录

附录一　常用的 C 库函数

1. 数学函数

这些函数包含在头文件“math. h”中。

使用数学函数时，应在该源文件中使用：#include "math. h"。

函数名	函数类型和形参类型	功能	返回值	说明
acos	double acos（x） double x;	计算 $\cos^{-1}$（x）的值	计算结果	x 应在-1 到 1 范围内
asin	double asin（x） double x;	计算 $\sin^{-1}$（x）的值	计算结果	x 应在-1 到 1 范围内
atan	double atan（x） double x;	计算 $\tan^{-1}$（x）的值	计算结果	
atan2	double atan2（x，y） double x，y;	计算 $\tan^{-1}$（x/y）的值	计算结果	
cos	double cos（x） double x;	计算 cos（x）的值	计算结果	x 单位为弧度
cosh	double cosh（x） double x;	计算 x 的双曲余弦 cosh（x）的值	计算结果	
exp	double exp（x） double x;	求 e^x的值	计算结果	
fabs	double fabs（x） double x;	求 x 的绝对值	计算结果	
floor	double floor（x） double x;	求出不大于 x 的最大整数	该整数的双精度实数	
fmod	double fmod（x，y） double x，y;	求出整除 x/y 的余数	返回余数的双精度数	
frexp	double frexp（val，eptr） double val; int *eptr;	把双精度数 val 分解为数字部分（尾数）x 和以 2 为底的指数 n，即 val = x * 2^n，n 存放在 eptr 指向的变量中	返回数字部分	

续表

函数名	函数类型和形参类型	功能	返回值	说明
log	double log（x） double x；	求 $\log_e x$，即 lnx	计算结果	
log10	double log10（x） double x；	求 $\log_{10} x$	计算结果	
modf	double modf（val，iptr） double valx；double *iptr；	把双精度数 val 分解为整数部分和小数部分，把整数部分存到 iptr 指向的单元中	val 的小数部分	
pow	double pow（x，y） double x，y；	计算 x^y 的值	计算结果	
sin	double sin（x） double x；	计算 sinx 的值	计算结果	x 的单位为弧度
sinh	double sinh（x） double x；	计算 x 的双曲正弦函数	计算结果	
sqrt	double sqrt（x） double x；	计算 x 的开方	计算结果	x≥0
tan	double tan（x） double x；	计算 tan（x） 的值	计算结果	x 的单位为弧度
tanh	double tanh（x） double x；	计算 x 的双曲正切函数值	计算结果	

2. 字符函数和字符串函数

ANSI C 标准要求在使用字符串函数时要包含头文件“string. h”，在使用字符函数时要包含头文件“ctype. h”。

函数名	函数类型和形参类型	功能	返回值	包含文件
isalnum	int isalnum（ch） int ch；	检查 ch 是否是字母或数字	是字母返回 1；否则返回 0	ctype. h
isalpha	int isalpha（ch） int ch；	检查 ch 是否是字母	是，返回 1 不是，返回 0	ctype. h
iscntrl	int iscntrl（ch） int ch；	检查 ch 是否为控制字符	是，返回 1 不是，返回 0	ctype. h
isdigit	int isdigit（ch） int ch；	检查 ch 是否为数字（0~9）	是，返回 1 不是，返回 0	ctype. h
isgraph	int isgraph（ch） int ch；	检查 ch 是否可打印字符（其 ASCII 码在 0x21 到 0x7E 之间），不包括空格	是，返回 1 不是，返回 0	ctype. h
islower	int islower（ch） int ch；	检查 ch 是否小写字母（a~z）	是，返回 1 不是，返回 0	ctype. h

续表

函数名	函数类型和形参类型	功能	返回值	包含文件
isprint	int isprint（ch） int ch;	检查 ch 是否可打印字符（其 ASCII 码在 0x20 到 0x7E 之间），包括空格	是，返回 1 不是，返回 0	ctype. h
ispunct	int ispunct（ch） int ch;	检查 ch 是否为标点字符（不包括空格），即除字母、数字和空格以外的所有可打印字符	是，返回 1 不是，返回 0	ctype. h
isspace	int isspace（ch） int ch;	检查 ch 是否为空格、跳格符（制表符）或换行符	是，返回 1 不是，返回 0	ctype. h
isupper	int isupper（ch） int ch;	检查 ch 是否为大写字母（A~Z）	是，返回 1 不是，返回 0	ctype. h
isxdigit	int isxdigit（ch） int ch;	检查 ch 是否为一个 16 进制数学字符（即 0~9，或 A~F，或 a~f）	是，返回 1 不是，返回 0	ctype. h
strcat	char *strcat（str1，str2） char *str1，*str2	把字符串 str2 接到 str1 后面，str1 最后的'\0' 被取消	str1	string. h
strchr	char *strchr（str，ch） char *str; int ch;	指向 str 指向的字符串中第一次出现 ch 的位置	返回指向该位置的指针，如找不到，则返回空指针	string. h
strcmp	int strcmp（str1，str2）char *str1，*str2;	比较两个字符串 str1、str2	str1<str2，返回负数； str1=str2，返回 0； str1>str2，返回正数	string. h
strcpy	char *strcpy（str1，str2） char *str1，*str2;	把 str2 指向的字符串拷贝到 str1 中去	返回 str1	string. h
strlen	unsigned int strlen（str） char *str;	统计字符串 str 中字符的个数（不包括终止符'\0'）	返回字符个数	string. h
strstr	char *strstr（str1，str2） char *str1，*str2;	找出 str2 字符串在 str1 字符串中第一次出现的位置（不包括 str2 的串结束符）	返回该位置的指针，如找不到，返回空指针	string. h

续表

函数名	函数类型和形参类型	功能	返回值	包含文件
tolower	int tolower（ch） int ch;	将 ch 字符转换为小写字母	返回 ch 所代表的字符的小写字母	ctype. h
toupper	int toupper（ch） int ch;	将 ch 字符转换为大写字母	返回 ch 所代表的字符的大写字母	ctype. h

3. 输入/输出函数

凡用以下的输入/输出函数，应该使用#include"stdio. h"把“stdio. h”头文件包含到源文件中。

函数名	函数类型和形参类型	功能	返回值	说明
clearerr	void clearerr（fp）file *fp;	清除文件指针错误	无	
close	int close（fp） int fp;	关闭文件	关闭成功返回 0，不成功返回-1	非 ANSI 标准
creat	int creat（filename，mode） char *filename; int mode;	以 mode 所指定方式建立文件	成功返回正数，否则返回-1	非 ANSI 标准
eof	int eof（fd） int fd;	检查文件是否结束	遇文件结束，返回 1，否则返回 0	非 ANSI 标准
fclose	int fclose（fp） FILE *fp;	关闭 fp 所指的文件，释放文件缓冲区	有错返回非 0，否则返回 0	
feof	int feof（fp） FILE *fp;	检查文件是否结束	遇文件结束，返回非零值，否则返回 0	
fgetc	int fgetc（fp） FILE *fp;	从 fp 所指定的文件中取得下一个字符	返回所得到的字符，若读入出错，返回 EOF	
fgets	char *fgets（buf，n，fp） char *buf; int n; FILE *fp;	从 fp 指向的文件读取一个长度为（n-1）的字符串，存入起始地址为 buf 的空间	返回地址 buf，若遇文件结束或出错，返回 NULL	

续表

函数名	函数类型和形参类型	功能	返回值	说明
fopen	FILE * fopen (filename, mode) char *filename, *mode;	以 mode 指定的方式打开名为 filename 的文件	成功，返回一个文件指针，否则返回 0	
fprintf	int fprintf (fp, format, args, ...) FILE *fp; char *format;	把 args 的值以 format 指定的格式输出到 fp 所指定的文件中	实际输出的字符数	
fputc	int fputc (ch, fp) char ch; FILE *fp	将字符 ch 输出到 fp 指定的文件中	成功，则返回该字符，否则返回 EOF	
fputs	int fputs (str, fp) char *str; FILE *fp	将 str 所指向的字符串输出到 fp 指定的文件中	成功返回 0，出错返回非 0	
fread	int fread (pt, size, n, fp) char *pt; unsigned size; unsigned n; FILE *fp;	从 fp 所指定的文件中读取长度为 size 的 n 个数据项，存到 pt 所指向的内存区	返回所读的数据项个数，如遇文件结束或出错返回 0	
fscanf	int fscanf (fp, format, args, ...) FILE *fp; char format;	从 fp 指定的文件中按 format 给定的格式将输入数据送到 args 所指向的内存单元	已输入的数据个数	
fseek	int fseek (fp, offset, base) FILE *fp; long offset; int base;	将 fp 所指向的文件的位置指针移到以 base 为基准，以 offset 为位移量的位置	返回当前位置，否则，返回-1	
ftell	long ftell (fp) FILE *fp;	返回 fp 所指向的文件中的读写位置	返回 fp 所指向的文件中的读写位置	
fwrite	int fwrite (ptr, size, n, fp) char *ptr; unsigned size; unsigned n; FILE *fp;	把 ptr 所指向的 n×size 个字节输出到 fp 所指向的文件中	写到 fp 文件中的数据项的个数	
getc	int getc (fp) FILE *fp;	从 fp 所指的文件中读入一个字符	返回所读的字符，若文件结束或出错，返回 EOF	

续表

函数名	函数类型和形参类型	功能	返回值	说明
getchar	int getchar（）	从标准输入设备读取下一个字符	返回所读字符，若文件结束或出错，返回-1	
getw	int getw（fp） FILE *fp;	从fp所指的文件中读取下一个整数	输入的整数。如文件结束或出错，返回-1	
open	int open（filename，mode） char *filename; int mode;	以mode指定的方式打开已存在的名为filename的文件	返回文件号（正数）。如打开失败，返回-1	
printf	int printf（format，args，...） char *format;	将输出表列args的值输出到标准输出设备	输出字符的个数，若出错，返回负数	format可以是一个字符串，或字符数组的起始地址
putc	int putc（ch，fp） int ch; FILE *fp;	把一个字符ch输出到fp所指的文件中	输出的字符ch，若出错，返回EOF	
putchar	int putchar（ch） char ch;	把字符ch输出到标准输出设备	输出的字符ch，若出错，返回EOF	
puts	int puts（str） char *str;	把str指向的字符串输出到标准输出设备，将'\0'转换成回车换行	返回换行符，若出错，返回EOF	
putw	int putw（w，fp） int w; FILE *fp;	将一个整数w（即一个字）写到fp指向的文件中	返回输出的整数，若出错，返回EOF	非ANSI标准函数
read	int read（fd，buf，count） int fd; char *buf; unsigned count;	从文件号fd所指示的文件中读count个字节到由buf指示的缓冲区中	返回真正读入的字节个数，如遇文件结束返回0，出错返回-1	非ANSI标准函数
rename	int rename（oldname，newname） char *oldname，*newname;	把由oldname所指的文件名，改为由newname所指的文件名	成功返回0，出错返回-1	

续表

函数名	函数类型和形参类型	功能	返回值	说明
rewind	void rewind（fp） FILE ＊fp;	将 fp 指示的文件中的位置指针置于文件开头位置，并清除文件结束标志和错误标志	无	
scanf	int scanf（format，args，...） char ＊format;	从标准输入设备按 format 指向的格式字符串规定的格式，输入数据给 args 所指向的单元	读入并赋给 args 的数据个数。遇文件结束返回 EOF，出错返回 0	args 为指针
write	int write（fd，buf，count） int fd; char ＊buf; unsigned count;	从 buf 指示的缓冲区输出 count 个字符到 fd 所标志的文件中	返回实际输出的字节数，出错返回-1	非 ANSI 标准函数

4. 动态存储分配函数

ANSI 标准建议设 4 个有关的动态存储分配的函数，即 calloc（）、malloc（）、free（）、realloc（）。实际上，许多 C 语言编译系统实现时，往往增加一些其他函数。ANSI 标准建议在“stdlib. h”头文件中包含有关的信息，但许多 C 编译要求采用“malloc. h”而不是“stdlib. h”。

函数名	函数类型和形参类型	功能	返回值	说明
calloc	void（或 char）＊calloc（n，size） unsigned n; unsigned size;	分配 n 个数据项的内存连续空间，每个数据项的大小为 size	分配内存单元的起始地址，如不成功，返回 0	
free	void free（p） void（或 char）＊p;	释放 p 所指的内存区	无	
malloc	void（或 char）＊malloc（n，size） unsigned n; unsigned size;	分配 size 字节的存储区	所分配的内存区地址，如内存不够，返回 0	
realloc	void（或 char）＊realloc（p，size） void（或 char）＊p; unsigned size;	将 f 所指出的已分配内存区的大小改为 size。size 可以比原来分配的空间大或小	返回指向该内存区的指针	

附录二　C 语言常见错误分析

1. 第一类错误分析

（1）在使用变量前未定义。

例如：

```
main( )
{a=1;
    b=2;
    printf("%d\n",a+b);
}
```

（2）语句后面漏写分号或不该加分号的地方加了分号。

C 语言规定，语句必须以分号结束，分号是 C 语句不可缺少的一部分，这也是和其他高级语言不同的一点。初学者往往容易忽略这个分号。如：

```
x=1
y=2;
```

又如在复合语句中漏写最后一个语句的分号：

```
{t=x;
    x=y;
    y=t
}
```

（3）不该有空格的地方加了空格。

例如，在用/*...*/对 C 程序中的任何部分作注释时，/与*之间不应当有空格。又如，在关系运算符<=，>=，==和!=中，两个符号之间也不允许有空格。

（4）定义或引用数组的方式不对。

C 语言规定，在对数组进行定义或对数组元素进行引用时必须要用方括号（对二维数组或多维数组的每一维数据都必须分别用方括号括起来），以下写法都将造成编译时出错：

```
int a(10); int b[5,4];
printf("%d\n",
b[1+2,2]);
```

（5）混淆字符和字符串。

C 语言中的字符常量是由一对单引号括起来的单个字符，而字符串常量是用一对双引号括起来的字符序列。字符常量存放在字符型变量中，而字符串常量只能存放在字符型数组中。例如：假设已说明 num 是字符型变量，则以下赋值语句是非法的：

```
num="1";
```

（6）在引用数组元素或指针变量之前没对其赋初值。

例如：

```
main()
{int a[6],b;
    b=a[5];
    ⋮
}
```

```
main()
{int *ptr,i=1;
    *ptr=i
    ⋮
}
```

以上两个程序段在编译时均会出现警告信息。

（7）混淆数组名与指针变量。

在C语言中，数组名代表数组的首地址，它的值是一个常量，不能被修改。例如，在以下程序段中，用a++是不合法的。

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

（8）混淆不同类型的指针。

若有以下语句：

```
int *p1,a=1;
float *p2;
p1=&a;
```

则赋值语句p2=p1是非法的。

（9）混淆指针说明语句中的*号和执行语句中的*号。

设有以下说明语句：

```
int *p1,i=1;
```

则 *p1=&i; 是不合法的。

（10）误将函数形参和函数中的局部变量一起定义。例如：

```
fun(x,y)
float x,y,z;
{x++; y++; z=x+y;
    ⋮
}
```

（11）所调用的函数在调用前未定义。

```
main()
{float a=1.0,b=2.0,c;
    c=fun(a,b);
    ⋮
}
float fun(x,y)
float x,y;
{x++; y++;
    ⋮
}
```

（12）混淆结构体类型名和结构体变量名。

若定义了以下结构体类型 student：

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

则赋值语句：student. num = 199401；是错误的。

2. 第二类错误分析

（1）在用 scanf（）函数给普通变量输入数据时，在变量名前漏写地址运算符 &。如：

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

（2）在 scanf（）函数调用语句中，企图规定输入实型数据的小数位。如执行以下语句：

```
scanf("%6.2f",&a);
```

（3）输入数据时的数据形式与要求不符。

用 scanf（）函数输入数据时，必须注意要与 scanf 语句中的对应形式匹配。如：

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

若按以下形式输入数据：

```
2 4
```

是不合法的。数据 2 和 4 之间应当有逗号。

（4）输入、输出时的数据类型与所用格式说明符不匹配。

例如有以下说明语句：

```
int x=1; float y=2.5;
```

则运行时执行语句：

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

将给出与原意不符的结果（在 TURBO C 2.0 下运行）。

（5）混淆“=”和“==”。

在 C 语言中，“=”是赋值运算符，“==”是关系运算符。

（6）在不该出现分号的地方加了分号。例如：

```
if(x>y);
printf("x is larger than y.\n");
```

（7）对于复合语句，忘记加花括号。例如：

```
i=1; a=0;
while(i<=10)
a+=i; i++;
printf("a=%d\n",a);
```

（8）误把数组说明时所定义的元素个数作为最大下标值使用。

C 语言规定，引用数组元素时下标从 0 开始，即下标值的下限为 0，而下标的上限值是数组定义时元素个数减 1。

（9）在 switch 语句的各分支中未使用 break 语句。例如：

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

（10）混淆 break 语句和 continue 语句的作用。例如：

```
do
{scanf("%d",&x);
     if(x>0)break;
     printf("%d\n",x);
}while(x! =0);
```

（11）使用++或--运算符时易犯的错误。例如：

```
main()
{int a[5]={1,2,3,4,5},*p;
     p=a;
     printf("%d\n",*(p++));
      ⋮
}
```

（12）误解形参值的变化会影响实参的值。例如：

```
main()
{int a=1,b=3;
     swap(a,b);
     printf("a=%d,b=%d\n",a,b);
}
swap(x,y)
int x,y;
{int m;
     m=x; x=y; y=m;
}
```

原意想通过调用 swap（）函数使 a 与 b 的值互换，然而，从输出结果可知 a 和 b 的值并未进行交换。

3. 常见错误信息语句英文索引

Ambiguous operators need parentheses：不明确的运算需要用括号括起

Ambiguous symbol 'xxx'：不明确的符号

Argument list syntax error：参数表语法错误

Array bounds missing] in function main：缺少数组界限符“]”

Array bounds missing：丢失数组界限符

Array size too large：数组尺寸太大
Bad character in paramenters：参数中有不适当的字符
Bad file name format in include directive：包含命令中文件名格式不正确
Bad ifdef directive synatax：编译预处理 ifdef 有语法错误
Bad undef directive syntax：编译预处理 undef 有语法错误
Bit field too large：位字段太长
Call of non-function：调用未定义的函数
Call to function with no prototype：调用函数时没有函数的说明
Cannot modify a const object：不允许修改常量对象
Case outside of switch：漏掉了 case 语句
Case syntax error：case 语法错误
Code has no effect：代码不可能执行到
Compound statement missing {：分程序漏掉“ {”
Conflicting type modifiers：不明确的类型说明符
Constant expression required：要求常量表达式
Constant out of range in comparison：在比较中常量超出范围
Conversion may lose significant digits：转换时会丢失意义的数字
Conversion of near pointer not allowed：不允许转换近指针
Could not find file ' xxx' ：找不到 xxx 文件
Declaration missing;：说明缺少“;”
Declaration syntax error：说明中出现语法错误
Default outside of switch：Default 出现在 switch 语句之外
Define directive needs an identifier：定义编译预处理需要标识符
Division by zero：用零作除数
Do statement must have while：Do…while 语句中缺少 while 部分
Enum syntax error：枚举类型语法错误
Enumeration constant syntax error：枚举常数语法错误
Error directive：xxx：错误的编译预处理命令
Error writing output file：写输出文件错误
Expression syntax error：表达式语法错误
Extra parameter in call：调用时出现多余错误
File name too long：文件名太长
Function call missing)：函数调用缺少右括号
Fuction definition out of place：函数定义位置错误
Fuction should return a value：函数必须返回一个值
Goto statement missing label：goto 语句没有标号
Hexadecimal or octal constant too large：16 进制或 8 进制常数太大
Illegal character ' x' ：非法字符 x

Illegal initialization：非法的初始化
Illegal octal digit：非法的 8 进制数字
Illegal pointer subtraction：非法的指针相减
Illegal structure operation：非法的结构体操作
Illegal use of floating point：非法的浮点运算
Illegal use of pointer：指针使用非法
Improper use of a typedefsymbol：类型定义符号使用不恰当
In-line assembly not allowed：不允许使用行间汇编
Incompatible storage class：存储类别不相容
Incompatible type conversion：不相容的类型转换
Incorrect number format：错误的数据格式
Incorrect use of default：default 使用不当
Invalid indirection：无效的间接运算
Invalid pointer addition：指针相加无效
Irreducible expression tree：无法执行的表达式运算
Lvalue required：需要逻辑值 0 或非 0 值
Macro argument syntax error：宏参数语法错误
Macro expansion too long：宏扩展以后太长
Mismatched number of parameters in definition：定义中参数个数不匹配
Misplaced break：此处不应出现 break 语句
Misplaced continue：此处不应出现 continue 语句
Misplaced decimal point：此处不应出现小数点
Misplaced elif directive：不应编译预处理 elif
Misplaced else：此处不应出现 else
Misplaced else directive：此处不应出现编译预处理 else
Misplaced endif directive：此处不应出现编译预处理 endif
Must be addressable：必须是可以编址的
Must take address of memory location：必须存储定位的地址
No declaration for function 'xxx'：没有函数 xxx 的说明
No stack：缺少堆栈
No type information：没有类型信息
Non-portable pointer assignment：不可移动的指针（地址常数）赋值
Non-portable pointer comparison：不可移动的指针（地址常数）比较
Non-portable pointer conversion：不可移动的指针（地址常数）转换
Not a valid expression format type：不合法的表达式格式
Not an allowed type：不允许使用的类型
Numeric constant too large：数值常数太大
Out of memory：内存不够用

Parameter 'xxx' is never used：参数 xxx 没有用到
Pointer required on left side of ->：符号->的左边必须是指针
Possible use of 'xxx' before definition：在定义之前就使用了 xxx（警告）
Possibly incorrect assignment：赋值可能不正确
Redeclaration of 'xxx'：重复定义了 xxx
Redefinition of 'xxx' is not identical：xxx 的两次定义不一致
Register allocation failure：寄存器定址失败
Repeat count needs an lvalue：重复计数需要逻辑值
Size of structure or array not known：结构体或数组大小不确定
Statement missing：语句后缺少“;”
Structure or union syntax error：结构体或联合体语法错误
Structure size too large：结构体尺寸太大
Sub scripting missing]：下标缺少右方括号
Superfluous & with function or array：函数或数组中有多余的“&”
Suspicious pointer conversion：可疑的指针转换
Symbol limit exceeded：符号超限
Too few parameters in call：函数调用参数太少
Too many default cases：default 太多（switch 语句中一个）
Too many error or warning messages：错误或警告信息太多
Too many type in declaration：说明中类型太多
Too much auto memory in function：函数用到的局部存储太多
Too much global data defined in file：文件中全局数据太多
Two consecutive dots：两个连续的句点
Type mismatch in parameter xxx：数 xxx 类型不匹配
Type mismatch in redeclaration of 'XXX'：重定义的类型不匹配
Unable to create output file 'xxx'：无法建立输出文件 xxx
Unable to open include file 'xxx'：无法打开被包含的文件 xxx
Unable to open input file 'xxx'：无法打开输入文件 xxx
Undefined label 'xxx'：没有定义的标号 xxx
Undefined structure 'xxx'：没有定义的结构 xxx
Undefined symbol 'xxx'：没有定义的符号 xxx
Unexpected end of file in comment started on line xxx：从 xxx 行开始的注解尚未结束文件不能结束
Unexpected end of file in conditional started on line xxx：从 xxx 开始的条件语句尚未结束文件不能结束
Unknown assemble instruction：未知的汇编结构
Unknown option：未知的操作
Unknown preprocessor directive：'xxx'：不认识的预处理命令 xxx

Unreachable code：无路可达的代码
Unterminated string or character constant：字符串缺少引号
User break：用户强行中断程序
Void functions may not return a value：void 类型的函数不应有返回值
Wrong number of arguments：调用函数的参数数目错误
'xxx' not an argument：xxx 不是参数
'xxx' not part of structure：xxx 不是结构体的一部分
xxx statement missing（：xxx 语句缺少左括号
xxx statement missing)：xxx 语句缺少右括号
xxx statement missing;：xxx 缺少分号
'xxx' declared but never used：说明了 xxx 但没有使用
'xxx' is assigned a value which is never used：给 xxx 赋了值但未用过
Zero length structure：结构体的长度为零

附录三　ASCII 码表

ASCII 值	控制字符	ASCII 值	控制字符	ASCII 值	控制字符	ASCII 值	控制字符
0	NUT	32	(space)	64	@	96	、
1	SOH	33	!	65	A	97	a
2	STX	34	”	66	B	98	b
3	ETX	35	#	67	C	99	c
4	EOT	36	$	68	D	100	d
5	ENQ	37	%	69	E	101	e
6	ACK	38	&	70	F	102	f
7	BEL	39	,	71	G	103	g
8	BS	40	(	72	H	104	h
9	HT	41	)	73	I	105	i
10	LF	42	*	74	J	106	j
11	VT	43	+	75	K	107	k
12	FF	44	,	76	L	108	l
13	CR	45	-	77	M	109	m
14	SO	46	.	78	N	110	n
15	SI	47	/	79	O	111	o
16	DLE	48	0	80	P	112	p

续表

ASCII 值	控制字符	ASCII 值	控制字符	ASCII 值	控制字符	ASCII 值	控制字符
17	DCI	49	1	81	Q	113	q
18	DC2	50	2	82	R	114	r
19	DC3	51	3	83	X	115	s
20	DC4	52	4	84	T	116	t
21	NAK	53	5	85	U	117	u
22	SYN	54	6	86	V	118	v
23	TB	55	7	87	W	119	w
24	CAN	56	8	88	X	120	x
25	EM	57	9	89	Y	121	y
26	SUB	58	:	90	Z	122	z
27	ESC	59	;	91	[	123	{
28	FS	60	<	92	/	124	|
29	GS	61	=	93	]	125	}
30	RS	62	>	94	^	126	~
31	US	63	?	95	—	127	DEL

各缩写意义如下：

缩写	意义	缩写	意义	缩写	意义
NUL	空	VT	垂直制表	SYN	空转同步
SOH	标题开始	FF	走纸控制	ETB	信息组传送结束
STX	正文开始	CR	回车	CAN	作废
ETX	正文结束	SO	移位输出	EM	纸尽
EOY	传输结束	SI	移位输入	SUB	换置
ENQ	询问字符	DLE	空格	ESC	换码
ACK	承认	DC1	设备控制 1	FS	文字分隔符
BEL	报警	DC2	设备控制 2	GS	组分隔符
BS	退一格	DC3	设备控制 3	RS	记录分隔符
HT	横向列表	DC4	设备控制 4	US	单元分隔符
LF	换行	NAK	否定	DEL	删除

参 考 文 献

[1] 谭浩强. C 语言设计学习辅导 [M]. 北京：清华大学出版社，2010.

[2] 张玉生，朱苗苗，张书月. C 语言程序设计实训教程 [M]. 上海：上海交通大学出版社，2018.

[3] 秦玉平，马靖善，王丽君. C 语言程序设计学习与实验指导 [M]. 3 版. 北京：清华大学出版社，2017.

[4] 孙辉. C 语言程序设计实验指导与习题集 [M]. 石家庄：中国铁道出版社，2016.

[5] 陈维. C 语言程序设计实训教程 [M]. 北京：人民邮电出版社，2018.

C 语言程序设计项目化教程

(实训指导)

主　编　付　琳　梁英坚

西北工业大学出版社

西　安

目　　录

第一部分　C 语言程序设计实训指导

在学习 C 语言时，上机实训非常重要。通过实训，可以对 C 语言功能特征、语法规则、程序编写和运行等加深理解。通过上机调试程序，学生能及时发现程序编写中出现的错误并找到改正的方法，提高学生的编程能力和编程技巧，为后续课程打下坚实的基础。

实训 1　VC++ 6.0 环境的操作

一、实训目的

1. 熟悉 VC++ 6.0 编译环境。
2. 熟悉 C 程序的编辑、编译、链接和运行的过程。
3. 通过运行简单的 C 程序，初步了解 C 程序的特点。

二、实训准备

1. 复习 VC++ 6.0 集成环境的基本操作步骤。
2. 复习 VC++ 6.0 集成环境常用到的命令和快捷键。

三、实训内容

实训题目 1

1. 在“我的电脑”上新建一个文件夹，用于存放 C 程序。
2. 调试示例，在屏幕上显示一句“hello world!”。

```
#include <stdio.h>
void main()
{
    printf("hello world! \n");
}
```

运行结果：

```
hello world!
```

基本步骤：

（1）启动 VC++ 6.0。“开始”→“所有程序”→“Visual C++ 6.0”进入 VC++编程环境，如图 1-1 所示。

图 1-1　Visual C++ 6.0 开发环境窗口

（2）创建工程并输入源程序代码。选择“文件”菜单下的“新建”项，会出现一个如图 1-2 所示的选择界面。选择“Win32 Console Application”，输入工程名称，选择第一步创建的文件夹。

图 1-2　新建名为 ch1 的工程

（3）选择默认“一个空工程”，单击“完成”按钮，如图 1-3 所示，即将生成一个空的工程。

图 1-3 选择创建工程类型

(4) 在工程中新建 C 源程序文件，并输入源程序代码。“文件”→“新建”，在“文件”选项卡选择“C++ Source Files”，设置文件名为“1-1. c”，如图 1-4 所示，点击“确定”按钮。

图 1-4 创建名为 1-1. c 的源程序文件

（5）输入源程序代码。（注意：字符、标点都必须是英文半角状态，同时注意大小写。）

这时可以通过 Workspace 窗口中的 FileView 标签，看到 Source Files 文件夹下文件 1-1.c 已经被添加进去，如图 1-5 所示。

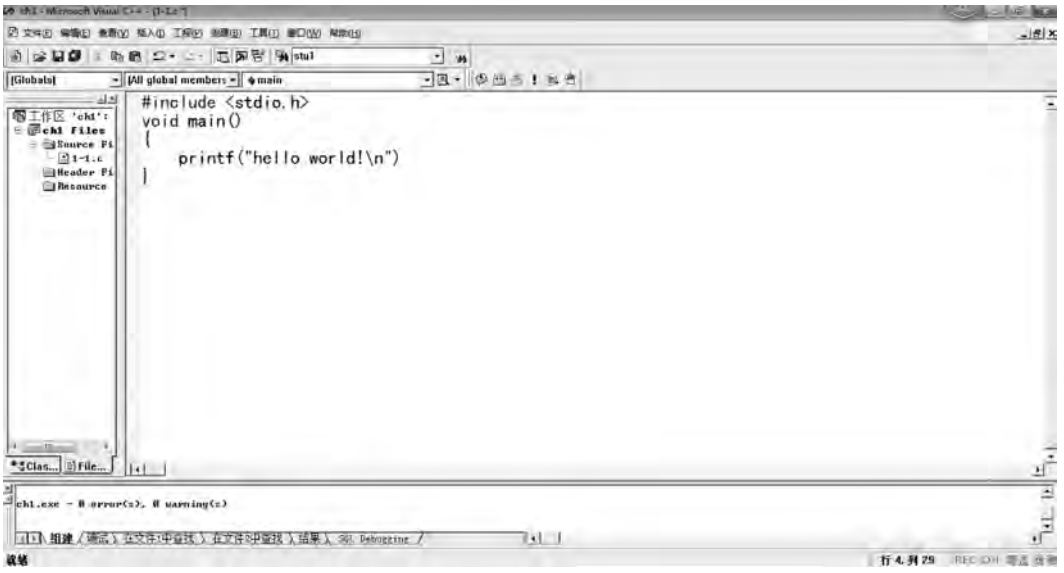


图 1-5　编辑源程序

（6）按 Ctrl+F7 进行编译，观察屏幕上显示的编译信息。出现出错信息“error C2143：syntax error：missing '；' before '｝'”，错误的解释为在“｝”前缺少“；”。双击错误提示，在编辑程序窗口有一个指针指向程序的第 5 行，如图 1-6 所示。在第 4 行末尾添加“；”后再次进行编译。

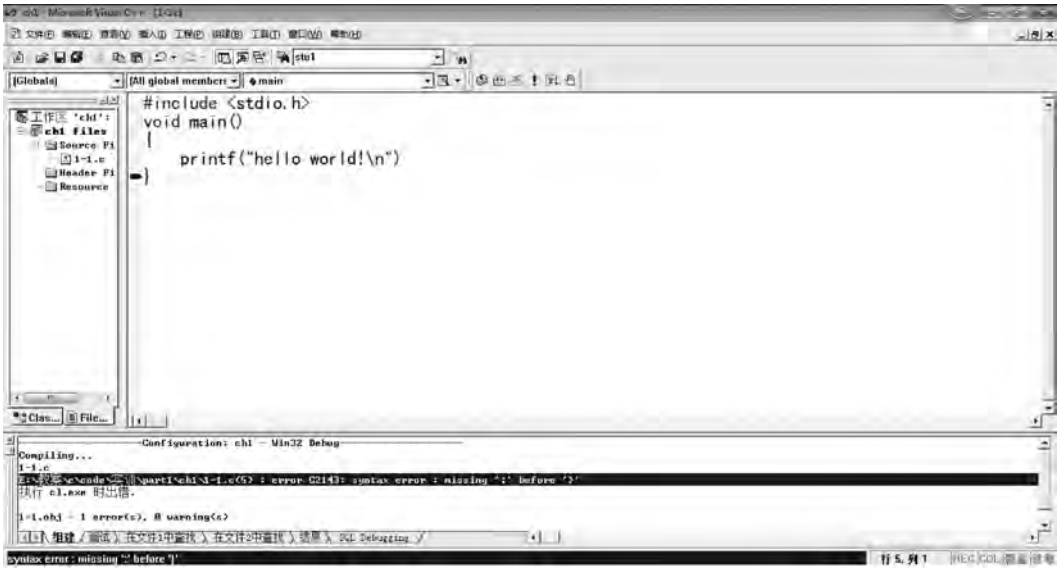


图 1-6　定位出错代码行界面

（7）按 F7 进行链接，观察屏幕上显示的链接信息。如果出现“出错信息”，则找出原因并改正，再进行链接。

（8）按 Ctrl+F5 运行程序，观察并分析运行结果。

实训题目 2

创建第 2 个程序“1-2. c”，输入以下程序：

```
#include <stdio. h>        /* 包含文件 */
void main( )
{
    int a,b,sum;                    /* e1 */
    a=10;b=20;                      /* e2 */
    sum=a+b;
    printf("sum=%d\ n", sum);
}
```

基本步骤：

（1）在工程中新建 C 源程序文件，并输入源程序代码。“文件”→“新建”，在“文件”选项卡选择“C++ Source Files”，设置文件名为“1-2. c”，如图 1-7 所示，点击“确定”按钮，即可在同一个工程中新建多个文件。

图 1-7 新建 1-2. c 文件

（2）在 Workspace 窗口中的 FileView 标签，可看到 Source Files 文件夹下文件 1-2. c 已经被添加进去。选择 1-1. c，单击鼠标右键，在快捷菜单中选择“设置”，出现如图 1-8

所示的对话框。在左边选择 1-1.c，在“常规”选项卡中勾选“组建时排除文件”。

图 1-8　设置文件

（3）输入上述程序后，保存该文件，编译、链接和运行程序，查看结果。退出 VC++ 6.0 集成环境，然后重新进入，试着调出刚存入的程序文件。

（4）把程序的 e2 行改成“scanf（"%d%d"，&a，&b）;”，重新运行，了解如何在运行期间向程序变量输入数据。

（5）把 e2 行改成“scanf（"%d,%d"，&a，&b）;”，试一试应如何输入数据。

（6）程序的 e1 行和 e2 行交换次序，编译会出现什么情况？请思考原因。

（7）程序中的大小写用错了，如 main 写成了 Main 或者 MAIN，结果会是怎么样的？

（8）一个程序中能否存在两个及两个以上的 main（）函数？

【分析与提示】

（1）程序运行结果：sum=30。

（2）把程序的 e2 行改成“scanf（"%d%d"，&a，&b）;”，按 Ctrl+F5 显示 DOS 界面，并且光标在闪烁，等待输入，输入 10 20<回车>，就显示“sum=30”。

（3）把 e2 行改成“scanf（"%d,%d"，&a，&b）;”，按 Ctrl+F5 显示 DOS 界面，并且光标在闪烁，等待输入，输入 10，20<回车>，同样显示 sum=30。

（4）重新运行，了解如何在运行期间向程序变量输入数据。

（5）程序中的大小写用错了，如 main 写成了 Main 或者 MAIN，结果会出现错误。

（6）一个程序中不能存在两个及两个以上的 main（）函数。

（7）程序运行成功后，按 Ctrl+S 或点击“文件→保存”保存工程。

（8）再次进入 VC++ 6.0 集成环境，按 Ctrl+O 或单击“文件”菜单下的打开命令来打开工程，双击“*.dsw”文件，即可调出刚存入的工程文件。

实训题目 3

创建第 3 个程序“1-3.c”，输入下面程序并运行。

```
#include <stdio.h>
void main()
{
    printf("******\n");
    printf ("*****\n");
    printf ("****\n");
    printf ("***\n");
    printf ("**\n");
    printf ("*\n");
}
```

要求：

（1）自己动手验证程序。

（2）修改程序，使之输出平行四边形和等腰三角形。

四、实训报告要求

将以上各题的源程序、运行结果，在实训中遇到的问题和解决问题的方法，以及实训心得体会填写在实训报告上。

实训 2　数据类型、运算符的使用

一、实训目的

1. 掌握 C 语言数据类型，掌握整型常量、字符型常量、实型常量、变量的定义方法和基本使用方法。

2. 学会使用 C 语言的基本赋值运算符以及复合赋值运算符。

3. 学会使用 C 语言的算术运算符，特别是自加（++）和自减（--）运算符。

4. 通过编写简单的 C 程序，了解 C 程序的编写方法和特点。

5. 进一步熟悉 C 程序编辑、编译、链接和运行的过程。

二、实训准备

1. 复习 C 语言数据类型，掌握对其进行定义和赋值的方法。

2. 复习不同数值型数据间的混合运算。

3. 复习 C 语言的运算符以及包含这些运算符的表达式的求值规则。

三、实训内容

实训题目 1

创建第 1 个程序“2-1.c”，输入并运行下面程序。

```
#include <stdio.h>
void main()
{
    int a,b,c;                          /* e1 */
    scanf("%d,%d", &a, &b);             /* e2 */
    c=a/b;
    printf ("%d\n", c);                 /* e3 */
}
```

要求：

调试通过后，运行该程序。

（1）输入：3，2<回车>，运行结果为：_____。

（2）输入：2.2，6.7<回车>，运行结果为：_____。

（3）输入：100000，80<回车>，运行结果为：_____。

（4）输入：5，0<回车>，运行结果为：_____。

【思考与编程】

（1）修改程序，将 e1 行改为：float a，b，c。

（2）将 e2、e3 行中的“%d”改为“%f”，运行程序，分析执行结果。

实训题目 2

创建第 2 个程序“2-2.c”，编写半径 r 为 10，求圆面积的程序。

参考代码：

```
#include<stdio.h>
#define PI 3.14159
void main()
{
    float r,area;                       //定义浮点型变量 r,area
    r=10.0;                             //为半径 r 赋值
    area=PI*r*r;                        //计算圆面积
    printf("面积为:", area);            //输出圆面积
}
```

说明：

程序第 2 行定义了符号常量 PI。这样做的好处是程序中出现的所有 PI，其值均为

3.141 59，既简化了程序，又提高了程序可读性。

实训题目 3

创建一个程序“2-3.c”，实现输入一个职工的姓名、年龄、工资的程序并输出。

参考代码：

```
#include <stdio.h>
voidmain()
{   char name[6];
    int  age;
    float wage;
    printf("请输入职工姓名:");
    scanf ("%s", name);
    printf ("请输入职工年龄:");
    scanf ("%d", &age);
    printf ("请输入职工工资:");
    scanf ("%f", &wage);
    printf ( “此职工 姓名:%s 年龄:%d 工资:%0.2f\n", name, age, wage);
}
```

要求：

分别输入姓名、年龄和工资，运行结果为：_____。

实训题目 4

创建一个程序“2-4.c”，输入并运行以下程序，写出注释和运行结果。

```
#include <stdio.h>
void main()
{
    int a,b,c;                    /*          */
    printf("请输入 a, b:");
    scanf ("%x,%x", &a, &b);          /*          */
    c=a+b;
    printf ("%d,%o,%x\n", c, c, c);     /*          */
}
```

说明：

（1）“%0”表示从键盘上输入八进制数据，输入时各位数字只能在 0~7 范围内。“%x，表示从键盘上输入十六进制数据，输入时各位数字只能在 0~9、a~f 范围内。

（2）运行程序，输入“12，1a”，输出的结果为：_____。

实训题目 5

创建一个程序“2-5.c”，实现输入一个三位正整数，输出其反序数（如 123 反序数为 321），请将程序完善。

```
#include <stdio.h>
void main()
{
    __(1)__;                                    /*定义变量*/
    num=123;
    a=__(2)__;
    b=(num-a*100)/10;                           /*计算百位数*/
    c=__(3)__;                                  /*计算个位数*/
    printf("反序数是:%d%d%d\n", c, b, a);       /*输出反序数*/
}
```

运行结果：

实训题目 6

创建一个程序“2-6.c”，输入以下程序，按要求写出结果。

```
#include <stdio.h>
void main()
{
    int i,j,m,n;
    i=8;j=10;
    m=++i;
    n=j++;
    printf("%d,%d,%d,%d\n", i, j, m, n);
}
```

运行结果：

将程序改为：

```
#include <stdio.h>
void main()
{
    int i,j;
    i=8;j=10;
    printf("%d,%d\n", i++, j++);
    printf ("%d,%d\n", ++i, j++);
}
```

运行结果：

实训题目 7

修改程序“2-7.c”，使运行结果为 y=9。请改正程序中的错误。（不允许增加或删除语句）

有错误代码：

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

```
void main( )
{
    int x,y;
    x=50;
    y=x=x-5,x/5;
    printf("y=%d\n", y);
}
```

运行结果：

```
y=9
```

实训题目 8

编写程序“2-8.c”，从键盘上输入一个小写字母，用大写字母输出。

【分析与提示】小写字母的 ASCII 码值比对应的大写字母的 ASCII 码值大 32。

参考代码：

```
#include <stdio.h>
void main( )
{
    char a;
    printf("请输入一个小写字母:");
    a=getchar ( );
    a=a-32;      /*将小写字母转换成对应的大写字母*/
    printf ("%c\n", a);
}
```

【思考与编程】

（1）如果使用 scanf（）输入、putchar（）输出，需要如何修改程序？

（2）从键盘上输入一个大写字母，转换为用小写字母输出，需要如何修改程序？

实训题目 9

创建程序“2-9.c”。

```
#include <stdio.h>
void main( )
{
    int a=300,b=200;
    long c;
    c=(long)a*b;
    printf("%d*%d=%ld/n", a, b, c);
}
```

【思考与编程】能否将 c 定义成 int 型？若将赋值语句改写成“c=a*b;”或“c=(long)(a*b)”，对吗？

四、实训报告要求

将以上各题的源程序、运行结果，在实训中遇到的问题和解决问题的方法，以及实训心得体会填写在实训报告上。

实训 3　顺序结构和选择结构

一、实训目的

1. 掌握 C 语言中基本语句的使用及顺序程序设计的一般方法。
2. 掌握输入函数和输出函数的用法。
3. 正确使用各种输入/输出格式。
4. 掌握 if 语句、switch 语句和条件运算符的格式与应用。
5. 能熟练地使用 if 语句、switch 语句进行选择结构的程序设计。

二、实训准备

1. 复习 C 语言的基本结构和基本数据类型。
2. 复习输入/输出函数（getchar（）、putchar（）、scanf（）、printf（））的调用格式与功能。
3. 复习 if 语句、switch 语句的用法。
4. 复习 VC++ 6.0 的基本操作方法，学会保存、修改源程序。

三、实训内容

实训题目 1（3-1.c）

分析下面程序的运行结果，并上机予以验证。

```
#include <stdio.h>
void main()
{
    int a=65,b=97;
    printf("%d,%d\n", a, b);
    printf ("%3d,%3d\n", a, b);
    printf ("%3c,%3c\n", a, b);
}
```

实训题目 2（3-2.c）

分析下面程序的运行结果，并上机予以验证。

```
#include <stdio.h>
void main()
{
    int a,b;
    float c,d;
    long e,f;
    unsigned int u,v;
    char c1,c2;
    scanf("%d,%d", &a, &b);
    printf ("a=%5d, b=%5d\n", a, b);
    scanf ("%f,%f", &c, &d);
    printf ("c=%5.2f, d=%5.2f\n", c, d);
    scanf (" %ld,%ld", &e, &f);
    printf ("e=%15ld, f=%15ld\n", e, f);
    scanf (" %o,%o", &u, &v);
    printf ("u=%o, v=%o\n", u, v);
    scanf (" %c,%c", &c1, &c2);
    printf ("c1=%d, c2=%d\n", c1, c2);
}
```

【分析与提示】

（1）当有多个输入函数 scanf（）时，从第二个开始，在第一个打印格式前需要加一个空格，用以吸收上一个输入结束时的回车，否则输入的数据会与所对应的变量错乱，从而导致错误的结果。

（2）调试上述程序至无语法错误后，用下面的数据对程序进行测试：

a=23，b=45

c=12.3，d=1234.567

e=65230，f=452688

u=45100，v=13485

c1='a'，c2='b'

分析运行结果。特别注意 c1、c2 的输出值是什么，为什么？

原因：v 按八进制数输入，最大值不能超过 7，所以输入 v=13485 是错误的，这会影响到 c1、c2 的值。如改为“v=13455”，c1、c2 的输出结果就会正确。

（3）将 e 和 f、u 和 v 的输入语句改为：

```
scanf("%d,%d", &e, &f);
scanf ("%d,%d", &u, &v);
```

再用上述数据测试并分析结果。

【思考与编程】

用 getchar（）函数读入两个字符给 c1 和 c2，用 putchar（）函数输出 c1 和 c2。分析

printf（）和 putchar（）函数输出字符的特点。

实训题目 3（3-3.c）

分析下面程序的运行结果，并将注释补齐。

```
#include <stdio.h>
void main()
{
    float x=7.12;
    int a=5,b;
    b=(int)x%a;     /*                  */
    printf("b=%d, x=%f\n", b, x);
}
```

运行结果为：_____。

实训题目 4（3-4.c）

任意输入 3 个实数，编程实现从小到大输出。

```
#include <stdio.h>
void main()
{
    float a,b,c,t;  /*定义3个变量,t为中间变量*/
    scanf("%f,%f,%f", &a, &b, &c);
    if (a>b)
      {t=a; a=b; b=t;}    /*使a小于b*/
    if (a>c)
      {t=a; a=c; c=t;}    /*使a小于c*/
    if (b>c)
      {t=b; b=c; c=t;}    /*使b小于c*/
    printf ("%5.2f,%5.2f,%5.2f\n", a, b, c);
}
```

实训题目 5（3-5.c）

编写一个程序，判断输入的正整数是奇数还是偶数。

```
#include <stdio.h>
void main()
{
    int n;
    printf("请输入一个正整数:");
    scanf ("%d", &n);
    if (n%2==0)
    printf ("%d是一个偶数\n", n);
```

```
    else
    printf ("%d 是一个奇数\n", n);
}
```

实训题目 6（3-6.c）

编写一个程序，要求输入一个数字，输出一个对应的星期几的英文单词。

```
#include <stdio.h>
void main()
{
    int a;
    printf("请输入一个整数 (1-7):");
    scanf ("%d", &a);
    switch (a)
     {
        case 1: printf ("Monday\n"); break;
        case 2: printf ("Tuesday\n"); break;
        case 3: printf ("Wednesday\n"); break;
        case 4: printf ("Thursday\n"); break;
        case 5: printf ("Friday\n"); break;
        case 6: printf ("Saturday\n"); break;
        case 7: printf ("Sunday\n"); break;
        default: printf ("error\n");
    }
}
```

实训题目 7（3-7.c）

分析下面程序的运行结果。

```
#include <stdio.h>
void main()
{
    int x=1,y=0,a=0,b=0;
    switch(x)
    {
        case  1:
        switch(y)
        {
            case 0:  a++;  break;
            case 1:  b++;  break;
        }
        case  2:  a++;b++; break;
```

```
        case  3:  a++;b++;
    }
    printf("\na=%d, b=%d", a, b);
}
```

实训题目 8（3-8.c）

编写一个程序，根据输入的年份和月份，判断当月的天数。

```
#include <stdio.h>
void main()
{
    int y,m,d;
    printf("请输入年和月:");
    scanf ("%d,%d", &y, &m);
    switch (m)
     {
        case 1:
        case 3:
        case 5:
        case 7:
        case 8:
        case 10:
        case 12: d=31;    break;
        case 4:
        case 6:
        case 9:
        case 11: d=30;    break;
        case 2:
        if (y%400==0||y%4==0&&y%100!=0) //2 月须判断是否是闰年
        d=29;
        else
        d=28;
        break;
    }
    printf ("%d 年%d 月的天数是:%d", y, m, d);
}
```

实训题目 9（3-9.c）

下面程序的功能是：由键盘输入一个字符，若该字符为小写字母，则将其转换为大写字母；若该字符为大写字母，则将其转换为小写字母；否则将其转换为 ASCII 码表中该字符的下一个字符。请改正程序中的错误，并调试。

注意：改错时不允许增加及删除语句，只允许修改或移动语句的位置。

```
#include<stdio. h>
void main( )
{
    char c1,c2;
    printf(" \ n 请输入一个字符:");
    c1= getchar ( );
    if (c1>="a"&&c1<="z")    /* 判断是否小写字母 */
    c2=c1+32;
    else if (c1>=A | | c1<=Z)    /* 判断是否大写字母 */
    c2=c1-32;
    else
    c2=c1+1;            /* 转换为下一个字符 */
    putchar (c2);
}
```

四、实训报告要求

将以上各题的源程序、运行结果，在实训中遇到的问题和解决问题的方法，以及实训心得体会填写在实训报告上。

实训 4　循 环 结 构

一、实训目的

1. 熟练掌握 while、do…while、for 三种循环语句的应用。
2. 熟练掌握循环结构的嵌套。
3. 掌握 break 和 continue 语句的使用。

二、实训准备

1. 复习 while、do…while、for 语句的用法。
2. 复习 break 和 continue 语句的用法。

三、实训内容

实训题目 1（4-1. c）

用所学的三种循环结构编程实现 n 的阶乘。

【分析与提示】一个正整数的阶乘（factorial）是所有小于及等于该数的正整数的积，

并且 0 的阶乘为 1。自然数 n 的阶乘写作 $n!$。任何大于等于 1 的自然数 n 的阶乘表示方法：$n! = 1\times2\times3\times\cdots\times n$。注意临界值的确定，不能多乘或者少乘。

（1）使用 while 语句实现，参考代码：

```
#include<stdio.h>//while
void main()
{
    int i,n,sum=1;
    printf("\n请输入一个整数 n:");
    scanf ("%d", &n);
    i=1;
    while (i<=n)
     {
        sum=sum*i;
        i++;
    }
    printf ("%d\n", sum);
}
```

（2）使用 do…while 语句实现，参考代码：

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

（3）使用 for 语句实现，参考代码：

```
#include<stdio.h>     //for
void main()
{
    int i,n,s=1;
    printf("\n请输入一个整数 n:");
```

```
    scanf ("%d", &n);
    for (i=1; i<=n; i++)
    s=s*i;
    printf ("%d\n", s);
}
```

实训题目 2（4-2.c）

编写一个程序，打印输出以下图形：

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

【分析与提示】利用双重 for 循环，外循环控制行，内循环控制列。内循环控制星号的输出。

```
#include<stdio.h>
void main()
{
    int i,j;
    for(i=1;i<=4;i++)
    {
        for(j=1;j<=2*i-1;j++)
        printf("*");
        printf ("\n");
    }
}
```

【思考与编程】

如何修改程序，打印输出以下图形：

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

【分析与提示】利用双重 for 循环，外循环控制行，内循环控制列。内循环又有两个并列的 for 循环，一个控制空格的输出，一个控制星号的输出。

实训题目 3（4-3.c）

编写一个程序，输出 1~200 以内的素数，按 5 个一行输出。

【分析与提示】素数是指只能被 1 和它本身整除的数。判断素数的方法：用一个数依次去除以 2 到这个数减 1 之间的数，如果能被整除，则表明此数不是素数，反之是素数。

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

```
void main( )
{
    int m,i,n=0;//m 代表从 2 到 i 的被除数,n 代表素数个数
    for(i=1;i<=200;i++)     /*循环初值 i 从 1 到 200*/
    {
        for(m=2;m<i;m++)
        if(i%m==0)        /*若有一个能除尽,则终止循环*/
        break;
        if(m>=i)                    /*若是素数应 m>=i   */
        {
            printf("%d\t", i);
            n++;
            if (n%5==0)
            printf ("\n");
        }
    }
    printf ("\n100-200 一共有%d 个素数", n);
}
```

实训题目 4（4-4.c）

求某一范围内完数的个数。

如果一个数等于它的因子（因子就是所有可以整除这个数的数）之和，则称该数为“完数”（或“完全数”）。例如，6 的因子为 1、2、3，而 6=1+2+3，因此 6 是“完数”。

【分析与提示】根据完数的定义，解决本题的关键是计算出所选取的整数 i（i 的取值范围不固定）的因子，将各因子累加到变量 s（记录所有因子之和），若 s 等于 i，则可确认 i 为完数，反之则不是完数。

```
#include<stdio.h>
int main( )
{
    int i,j,s,n;  /*变量 i 控制选定数范围,j 控制除数范围,s 记录累加因子之和*/
    printf("请输入所选范围上限:");
    scanf ("%d", &n);   /* n 的值由键盘输入*/
    for (i=2; i<=n; i++)
     {
        s=0;   /*保证每次循环时 s 的初值为 0*/
        for (j=1; j<i; j++)
         {
            if (i%j == 0)   /*判断 j 是否为 i 的因子*/
            s += j;
```

```
        }
        if (s == i)    /*判断因子的和是否和原数相等*/
        printf ("该范围内的完数为:%d\n", i);
    }
    return 0;
}
```

实训题目 5（4-5.c）

求任意两个正整数的最大公约数。

【分析与提示】如果有一个自然数 a 能被自然数 b 整除，则称 a 为 b 的倍数，b 为 a 的约数。几个自然数公有的约数，叫作这几个自然数的公约数。公约数中最大的一个公约数，称为这几个自然数的最大公约数。

根据约数的定义可知，某个数的所有约数必不大于这个数本身，几个自然数的最大公约数必不大于其中任何一个数。要求任意两个正整数的最大公约数即求出一个不大于其中两者中的任何一个，但又能同时整除两个整数的最大自然数。

```
#include<stdio.h>
int main()
{
    int m,n,temp,i;
    printf("Input m & n:");
    scanf ("%d%d", &m, &n);
    if (m<n)    /*比较大小，使得 m 中存储大数，n 中存储小数*/
     {/*交换 m 和 n 的值*/
        temp=m;
        m=n;
        n=temp;
    }
    for (i=n; i>0; i--)    /*按照从大到小的顺序寻找满足条件的自然数*/
    if (m%i==0 && n%i==0)
     {/*输出满足条件的自然数并结束循环*/
        printf ("The GCD of %d and %d is:%d\n", m, n, i);
        break;
    }
    return 0;
}
```

实训题目 6（4-6.c）

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

【分析与提示】在 10 万以内判断，先将该数加上 100 后开一次方，再将该数加上 168 后开二次方，如果开二次方后的结果满足条件，即是所求整数。本题需要在 10 万以内找数字，显然要用到循环。

参考代码：

```
#include <stdio.h>
#include <math.h>
void main()
{
    long i,x,y,z;
    for(i=1;i<100000;i++)
    {
        x=sqrt(i+100);
        y=sqrt(i+168);
        if((x*x==i+100)&&(y*y==i+168))
        printf("\n%ld\n", i);
    }
}
```

实训题目 7（4-7.c）

打印出所有“水仙花数”。所谓“水仙花数”，是指一个三位数，其各位数字三次方和等于该数本身。例如：153 是一个水仙花数，因为 $153=1^3+5^3+3^3$。

```
#include <stdio.h>
#include <math.h>
void main()
{
    int x=100,a,b,c;
    while(x>=100&&x<1000)
    {
        a=0.01*x;
        b=10*(0.01*x-a);
        c=x-100*a-10*b;
        if(x==(pow(a,3)+pow(b,3)+pow(c,3)))
        printf("%5d", x);
        x++;
    }
}
```

四、实训报告要求

将以上各题的源程序、运行结果，在实训中遇到的问题和解决问题的方法，以及实训

心得体会填写在实训报告上。

实训 5　函　　数

一、实训目的

1. 掌握函数的定义、说明和调用方法。
2. 掌握函数的参数及其传递方式，函数值的正确返回。

二、实训准备

1. 复习函数的定义和调用方法。
2. 复习函数参数的传递方式。
3. 复习函数值的正确返回。

三、实训内容

实训题目 1（5-1.c）

运行以下示例程序，查看运行结果。

```
#include<stdio.h>
int main()
{
    void p1();  //对 p1()函数进行声明
    void p2();  //对 p2()函数进行声明
    p1();       //对 p1()函数调用
    p2();       //对 p2()函数调用
    p1();       //再次对 p1()函数调用
    return 0;
}
void p1()                   //定义 p1()函数
{
    printf("* * * * * * * * * * * * * * * * * * * * * * * * * * * * * \n");
}
void p2 ()                  //定义 p2 () 函数
{
    printf ("你好           \n");
}
```

实训题目 2（5-2.c）

设计一个函数，用于判断一个数是否为素数。

```
#include <stdio.h>
#include <math.h>
int IsPrime(int num)
{
    int flag = 1,i;
    for(i = 2; i<=sqrt(num);i++)
    if(num%i == 0)
    {
        flag = 0;
        break;}
        return flag;
    }
    void main()
{
    int n;
    printf("请输入一个正整数:");
    scanf ("%d", &n);
    if (IsPrime (n) ==1)
    printf ("%d是一个素数", n);
    else
    printf ("%d不是一个素数", n);
}
```

实训题目 3（5-3.c）

下列程序的功能是寻找 10 000 以内具有下列特性的四位正整数：其百位数为 0，去掉百位数 0 可得到一个三位正整数，而该三位正整数乘以 9 等于原四位数。例如 6 075 = 675×9，请补齐代码：

```
#include<stdio.h>
int fun(int n)
{
    int a,b,c,d;
    a=n/1000;
    b=_____ ;                    //求百位数
    c= _____;                    //求十位数
    d=n%10;
    if((b==0)&&n==(_____)        //判断是否满足
    return 1;
```

```
    else
    return 0;
}
void main( )
{
    int i;
    for( i = 1000;i<10000;i++)
    if( _____ )                          //判断 i 是否是要找的数
    printf( "%6d", i);
    printf ("\n");
}
```

实训题目 4（5-4.c）

用递归的方法求 1+2+3+…+n 的和。

```
#include <stdio. h>
void main( )
{
    int sum( int);
    int n,s;
    printf( "输入 n 的值:");
    scanf ("%d", &n);
    s=sum (n);
    printf ("和为%d\n", s);
}
int sum (int n)
{
    if (n==0)                            //递归出口
    return 0;
    else
    return n+sum (n-1);
}
```

实训题目 5（5-5.c）

设计一个函数，计算 $s = 1! + 2! + 3! + \cdots + 10!$。

```
#include <stdio. h>
long fnFact( int n)
{
    int i;   long f=1L;
    for( i=1; i<=n; i++)
    f=f*i;
```

```
    return f;
}
void fnSum(int n)
{
    int i;
    long s=0L;
    for(i=1; i<=n; i++)
    s += fnFact(i);                          //调用 fnfact()函数
    printf("1! + 2! + … +%d! =%d", n, s);
}
void main ()
{
    int num;
    printf ("请输入一个正整数 (<=12):");
    scanf ("%d", &num);
    fnSum (num);                             //调用 fnSum () 函数
}
```

实训题目 6（5-6.c）

斐波那契（Fibonacci）数列为 1，1，2，3，5，8，13，用递归法编写求 Fibonacci 数列的函数，在主函数中输入一个自然数，输出不小于该自然数的最小的一个 Fibonacci 数列。请补齐下面的程序。

```
#include<stdio.h>
#include<math.h>
int fib(int n)
{
    int f;
    if(_____)                                //递归出口
    f=1;
    else
    f=_____;                                 //递归
    return f;
}
void main()
{
    int i,m,jieguo;
    printf("请输入一个数:");
    scanf ("%d", &m);
    i=1;
    while (_____)                            //找出大于 m 的数
```

```
    i++;
    printf ("找到大于该数的第一个斐波那契数列为%d\n", jieguo);
}
```

实训题目 6（5-6.c）

编写一个计算圆面积的函数。

```
#include <stdio.h>
#include<math.h>
#define PI 3.1415926
double syuan(double r)
{
    double s;
    //s=r*r*3.14;
    s=PI*pow(r,2);
    return s;
}
void main()
{
    float a;
    printf("请输入圆的半径:");
    scanf ("%f", &a);
    printf ("面积为%.2lf", s_yuan (a) );
}
```

四、实训报告要求

将以上各题的源程序、运行结果，在实训中遇到的问题和解决问题的方法，以及实训心得体会填写在实训报告上。

实训 6　数　　组

一、实训目的

1. 熟练掌握一维和二维数组的说明和使用方法。
2. 熟练掌握数组的常用算法。
3. 掌握字符数组的使用方法。
4. 熟练运用字符串库函数处理字符串。

二、实训准备

1. 复习一维和二维数组的定义、赋值和使用方法。
2. 复习字符数组的定义、赋值和使用方法。
3. 复习字符串库函数的功能及使用方法。

三、实训内容

实训题目 1（6-1.c）

从键盘输入 10 个整数，找出其中的最小值，并显示出来。

【分析与提示】成批处理多个数据，用数组是最便捷的方式，结合循环，让数组中所有数和当前最小的数值进行比较。

参考代码：

```
#include <stdio.h>
void main()
{
    int a[10],i,min;
    printf("请输入十个数字:");
    for (i=0; i<10; i++)
    scanf ("%d", &a [i] );
    min=a [0];
    for (i=1; i<10; i++)
    if (a [i] <min)
    min=a [i];
    printf ("最小值是:%d", min);
}
```

【思考与编程】

（1）如果找一维数组的最大值，如何改写程序？

（2）设计一个表演竞赛现场评分小程序，按计分规则现场统计评委评的分数，然后计算平均得分（去掉最高分和最低分），打印输出某选手的得分（假设有 10 个评委）。

实训题目 2（6-2.c）

求出 Fibonacci 数列的前 20 项，并存储在数组中，然后再按每行 5 个数据输出。

参考代码：

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

```
    for(i=2;i<20;i++)
    {
        a[i] = a[i-1]+a[i-2];
    }
    for(i=0;i<20;i++)
    if(i%5==0)
    printf("\n%5d", a[i]);
    else
    printf ("%5d", a[i]);
}
```

实训题目 3（6-3.c）

建立一个二维数组，求其横向数值之和并显示。例如：

a	[3]	[4]
10	20	30
40	50	60
70	80	90

计算后：

a	[3]	[4]	=
10	20	30	60
40	50	60	150
70	80	90	240

【分析与提示】成批处理二维数组数据运用双重循环，把行坐标相同、列坐标从 0 到 2 的数组元素相加，存储到同行、列坐标是 3 的元素中。

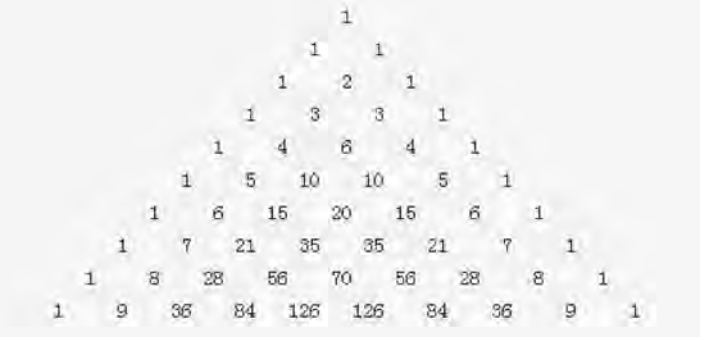
参考代码：

```
#include <stdio.h>
void main()
{
    int a[3][4]={{10,20,30},{40,50,60},{70,80,90}};
    int i,j;
    for(i=0;i<3;i++)
    for(j=0;j<3;j++)
    a[i][3]=a[i][3]+a[i][j];
    printf("计算后的二维数组:\n");
    for (i=0; i<3; i++)
     {
        for (j=0; j<4; j++)
        printf ("%4d", a[i][j]);
        printf ("\n");
```

```
    }
}
```

实训题目 4（6-4.c）

打印 8 行杨辉三角形。左右对称，由 1 开始逐渐增大，然后变小。第 n 行数字个数为 n 个，每个数字等于上一行的左右两个数字之和。10 行杨辉三角如下：



参考代码：

```
#include <stdio.h>
#define   N      8
#define   M      8
int   a[N][M]={0};
void main()
{
    int i,j;
    for(i=0;i<N;i++)
    a[i][0] = 1;
    for(i=1;i<N;i++)
    for(j=1;j<i+1;j++)
    a[i][j] = a[i-1][j-1]+a[i-1][j];
    for(i=0;i<N;i++)
    {
        for(j=0;j<M-i;j++)
        printf("   ");
        for (j=0; j<=i; j++)
        printf ("%4d", a [i] [j] );
        printf ("\n");
    }
}
```

实训题目 5（6-5.c）

输入一个字符串，将其中大写字母变成小写字母。

【分析与提示】运用循环提取出字符串中的每个字符，并判断其是否为大写字母，若是，则其值加 32。这是因为在 C 语言中，字符型数据是可以和整型数据进行运算的，而字母的大小写 ASCII 码值相差 32。

参考代码：

```
#include <stdio.h>
void main()
{
    char s[40];
    char c;
    int i;
    printf("请输入字符串:");
    gets (s);
    printf ("原字符串:");
    puts (s);
    for (i=0; (c=s [i] )! ='\0'; i++)
    if (c>=65 && c<=90) s [i] =s [i] +32;
    s [i] ='\0';
    printf ("更改后的字符串:");
    puts (s);
}
```

【思考与编程】

输入一个字符串，如何把小写字母转变成大写字母。

实训题目 6（6-6.c）

若有三个字符串 s1、s2 和 s3，实现以下功能：将 s1 的内容复制到 s3 中，并将 s2 中的内容添加在 s3 的后面，最后输出字符串 s3。

【分析与提示】先把 s1 字符串中的字符运用循环依次赋值给字符数组 s3，同理再把 s2 字符串中的字符赋值给字符数组 s3 已有字符后面的元素。需要注意的是，把 s2 赋给 s3 时，两个数组的下标变化是不同的，s2 是从 0 开始，而 s3 是从 s1 的元素个数开始。

参考代码：

```
#include <stdio.h>
void main()
{
    char s1[20],s2[20],s3[40];
    int i,j;
    printf("请输入字符串 1:");
    gets (s1);
    printf ("请输入字符串 2:");
    gets (s2);
```

```
    for (i=0; s1 [i]! ='\0'; i++)
    s3 [i] =s1 [i];
    for (j=i, i=0; s2 [i]! ='\0'; j++, i++)
    s3 [j] =s2 [i];
    s3 [j] ='\0';
    printf ("输出字符串 s3:");
    puts (s3);
}
```

实训题目 7（6-7.c）

输入若干个整数（值在 1~10 范围内），用 0 做结束标志，试编写一个函数用于统计每个整数出现的次数。

【分析与提示】在主函数中用一个一维数组存放一组数，用另一个一维数组存放每个数出现的次数，如数组 b 中用 b［0］存放 1 出现的次数，用 b［1］存放 2 出现的次数……将两个数组都传递给子函数，用子函数来判断每个数，并计算出现的次数。

参考代码：

```
#include <stdio. h>
#define M 50
f(int a[ ],int b[ ],int n)
{
    int i;
    for(i=0;i<n;i++)
    b[a[i]]++;
}
void main( )
{
    int a[M],b[11]={0},i,n,x;
    n=0;
    printf("please input (1~10), to end with 0\n");
    scanf ("%d", &x);
    while (x! =0)
     {
        if (x>=1&&x<=10)
         {
            a [n] =x;
            n++;
        }
        scanf ("%d", &x);
    }
```

```
    f (a, b, n);
    printf ("output the result: \n");
    for (i=1; i<=10; i++)
    printf ("%d:%d\n", i, b [i] );
    printf ("\n");
}
```

四、实训报告要求

将以上各题的源程序、运行结果，在实训中遇到的问题和解决问题的方法，以及实训心得体会填写在实训报告上。

实训 7　指　　针

一、实训目的

1. 理解指针的含义，能正确说明和使用指针。
2. 学会使用指向一维和二维数值型数组的指针解决实际问题。
3. 学会使用字符指针、指针数组。

二、实训准备

1. 复习指针的定义、赋值和使用方法。
2. 复习字符指针、指针数组的使用方法。

三、实训内容

实训题目 1 (7-1.c)

用指针编写冒泡排序的程序。

【分析与提示】定义一个普通的指针变量，让其依次指向数组中要进行比较的元素即可。

参考代码：

```
#include <stdio.h>
void main()
{
    int a[10], *p;
    int i,j,t;
    printf("请输入十个整数: \n");
    for (p=a; p<a+10; p++)
```

```
    scanf ("%d", p);
    printf ("\n");
    p=a;
    for (j=0; j<9; j++)
    for (i=0; i<9-j; i++)
    if ( * (p+i) > * (p+i+1) )
     {
        t= * (p+i);
        * (p+i) = * (p+i+1);
        * (p+i+1) =t;
    }
    printf ("排序后:\n");
    for (p=a; p<a+10; p++)
    printf ("%d ", *p);
}
```

实训题目2（7-2.c）

调试下面程序，使之具有如下功能：任意输入两个数，调用两个函数，分别求：①两个数的和；②两个数交换值。要求用函数指针调用这两个函数，结果在主函数中输出。

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

```
    t=a;
    a=b;
    b=t;
}
```

【分析与提示】调试程序时注意参数传递的是数值还是地址。

正确程序：

```
#include <stdio.h>
int sum(int a,int b);
int  swap(int *a,int *b);
void main()
{
    int a,b,c,(*p)();
    scanf("%d,%d", &a, &b);
    p=sum;
    c= (*p) (a, b);
    p=swap;
     (*p) (&a, &b);
    printf ("sum=%d\n", c);
    printf ("a=%d, b=%d\n", a, b);
}
int sum (int a, int b)
{
    return (a+b);
}
int swap (int *a, int *b)
{
    int t;
    t= *a;
     *a= *b;
     *b=t;
    return 0;
}
```

实训题目 3（7-3.c）

调试下面程序，使之具有如下功能：任意输入两个字符串（如:" abc 123" 和 "china"），存放在 a、b 两个数组中。然后把较短的字符串放入 a 数组，较长的字符串放入 b 数组，并输出。

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

```
{
    char a[10],b[10];
    int c,d,k;
    printf("Please input a string: \n");
    gets (a);
    printf ("Please input b string: \n");
    gets (b);
    c=strlen (a);
    d=strlen (b);
    if (c>d)
    for (k=0; a [k]! ='\0'; k++)
     { ch=a [k]; a [k] =b [k]; b [k] =ch;}
    printf ("最后结果为: \n");
    printf ("a=%s\n", a);
    printf ("b=%s\n", b);
}
```

【分析与提示】程序中的 strlen（）是库函数，功能是求字符串的长度，它的原型保存在头文件“string. h”中。调试时注意库函数的调用方法、不同的字符串输入方法，通过错误提示发现程序中的错误。

正确程序：

```
#include <stdio. h>
#include <string. h>
void main( )
{
    char a[10],b[10],ch;
    int c,d,k;
    printf("Please input a string: \n");
    gets (a);
    printf ("Please input b string: \n");
    gets (b);
    c=strlen (a);
    d=strlen (b);
    if (c>d)
     {
        for (k=0; a [k]! ='\0'; k++)
         { ch=a [k]; a [k] =b [k]; b [k] =ch;}
        b [k] ='\0';
    }
    printf ("最后结果为: \n");
```

```
    printf ("a=%s\n", a);
    printf ("b=%s\n", b);
}
```

实训题目 4（7-4.c）

编写一个程序，统计一个长度为 2 的字符串在另一个字符串中出现的次数。例如：假定输入的字符串为：asd asasdfg asd as zx67 asd mklo，子字符串为 as，则应输出 6。

【分析与提示】可以定义两个指针，第一个指向主串的首地址即 p=str，第二个指向子串的首地址即 r=substr，如果对应的字符相等（*r==*p），那么 r++，p++，即两个指针都向后移一个单位，再继续比较 *r 与 *p；否则 r 回到子串的首地址，只把指针 p 向后移一个单位，即 p++。依此类推，如果 *r=='\0'，说明子串已经终止，子串在主串中出现的次数加 1，直到主串终止为止。

参考代码：

```
#include <stdio.h>
#include <string.h>
#include <conio.h>
int fun(char *str,char *substr)
{
    int n;
    char *p,*r;
    n=0;
    while(*str)
    {
        p=str;
        r=substr;
        while(*r)
        if(*r==*p){r++;p++;}
        else break;
        if(*r=='\0')n++;
        str++;
    }
    return n;
}
void main()
{
    char str[81],substr[3];
    int n;
    printf("输入主字符串:");
    gets (str);
```

```
    printf ("输入子字符串:");
    gets (substr);
    n=fun (str, substr);
    printf ("n=%d\ n", n);
}
```

实训题目 5（7-5. c）

fun（）的功能是：计算形参 x 所指数组中 *N* 个数的平均值（规定全部为正数），将所指数组中大于平均值的数移到数组的前面，小于等于平均值的数移到数组的后面，平均值作为函数返回值。在主函数输出平均值和移动后的数据。

例如：46 40 32 40 6 17 45 15 48 26

平均值：30. 500000

移动后：46 32 40 45 48 30 6 17 15 26

请将下面程序补齐。

```
#include   <stdlib. h>
#include   <stdio. h>
#define   N   10
double fun(double   *x)
{ int   i,j;     double   s,av,y[N];
    s=0;
    for(i=0; i<N; i++)   s=s+x[i];
    /**********found**********/
    av= __1__ ;
    for(i=j=0; i<N; i++)
    if(x[i]>av){
    /**********found**********/
    y[ __2__ ]=x[i];
    x[i]=-1;}
    for(i=0; i<N; i++)
    /**********found**********/
    if(x[i]! = __3__ )y[j++]=x[i];
    for(i=0; i<N; i++)x[i] = y[i];
    return   av;
}
void main( )
{ int   i;     double   x[N];
    for(i=0; i<N; i++){ x[i]=rand( )%50; printf("%4. 0f ", x [i] );}
    printf ("\ n");
    printf ("\ nThe average is:%f\ n", fun (x) );
    printf ("\ nThe result: \ n", fun (x) );
    for (i=0; i<N; i++)    printf ("%5. 0f ", x [i] );
```

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

实训题目 6（7-6.c）

fun（）的功能是建立一个 $N \times N$ 阶的矩阵，矩阵元素的构成规律是：最外层的值全部是 1，从外向内第二层的值全部为 2，第三层的元素的值全部为 3……依此类推，假如 $N=5$，生成的矩阵为：

```
1 1 1 1 1
1 2 2 2 1
1 2 3 2 1
1 2 2 2 1
1 1 1 1 1
```

请将下面程序补齐。

```
#include   <stdio.h>
#define   N   7
/**********found**********/
void fun(int( *a) __1__ )
{ int   i,j,k,m;
    if(N%2==0)m=N/2 ;
    else        m=N/2+1;
    for(i=0; i<m; i++){
        /**********found**********/
        for(j= __2__ ; j<N-i; j++)
        a[i][j]=a[N-i-1][j]=i+1;
        for(k=i+1; k<N-i; k++)
        /**********found**********/
        a[k][i]=a[k][N-i-1]= __3__ ;
    }
}
void main()
{ int   x[N][N]={0},i,j;
    fun(x);
    printf("\nThe result is: \n");
    for (i=0; i<N; i++)
     {   for (j=0; j<N; j++)    printf ("%3d", x [i] [j] );
        printf ("\n");
    }
}
```

四、实训报告要求

将以上各题的源程序、运行结果，在实训中遇到的问题和解决问题的方法，以及实训心得体会填写在实训报告上。

实训 8　结构体与共用体

一、实训目的

1. 掌握结构体类型的定义、结构体变量的说明和引用。
2. 掌握结构体数组的定义及应用。
3. 掌握指针在结构体中的应用。
4. 了解共用体和枚举类型。

二、实训准备

1. 复习结构体类型的定义、结构体变量的定义与使用方法。
2. 复习结构体数组的定义与使用方法。
3. 复习共用体和枚举类型的定义与使用方法。

三、实训内容

实训题目 1（8-1.c）

求 Tom 的三门课程的成绩总分，并在显示器显示出来（使用结构型变量成员的引用）。

```
#include<stdio.h>
#include<string.h>
struct student
{ long number;
    char name[10];
    char sex;
    float score[3];
};
void main()
{
    struct student stu1;                          /*定义 student 类型的变量 stu1*/
    stu1.number = 100001L;                        /*给结构型变量 stu1 的成员 number 赋
值*/
    strcpy(stu1.name,"Tom");                      /*给结构型变量 stu1 的成员 name[] 赋
值*/
    stu1.sex='f';                                 /*给结构型变量 stu1 的成员 sex 赋值*/
    stu1.score[0] = 89;                           /*给结构型变量 stu1 的成员 score[] 赋
```

```
值*/
        stu1.score [1] =91;
        stu1.score [2] =86;
        printf ("%s 的总分是%f\n", stu1.name, stu1.score [0] +stu1.score [1] +stu1.score [2] );
    }
```

实训题目 2（8-2.c）

对候选人选票的统计程序。设有三个候选人，每次输入一个候选人的名字，要求最后统计出每人的得票结果。

```
#include <stdio.h>
#include <string.h>
struct person                                   /*定义描述候选人数据的结构体类型*/
{
    char name[20];                              /*存放候选人姓名*/
    int count;                                  /*存放得票数*/
    }leader[3]={"李军", 0,"张山", 0,"赵四", 0};   /*数组大小为3，有三个候选人*/
main ()
{
    int i, j;
    char name [20];
    printf ("请输入候选人名字（李军张山赵四）:\n");
    for (i=1; i<=10; i++)                       /*假设共10张票*/
     {
        printf ("%2d:", i);
        scanf ("%s", name);                     /*输入候选人的名字*/
        for (j=0; j<3; j++) /*每唱票1次，将相应候选人的票数加1*/
        if (! strcmp (strlwr (name), leader [j] .name) )
        leader [j] .count++;                    /*输入名与候选人名比较*/
    }
    printf ("\n");
    for (i=0; i<3; i++)       printf ("%5s:%d\t", leader [i] .name, leader [i] .count);
}
```

【分析与提示】

（1）在定义结构体类型时应预留出存放计算结果的成员项。

（2）用结构体数组名作函数参数，将各数据传给相应的函数。

四、实训报告要求

将以上各题的源程序、运行结果，在实训中遇到的问题和解决问题的方法，以及实训心得体会填写在实训报告上。

实训9　文　　件

一、实训目的

1. 熟练掌握文件的打开、关闭函数。
2. 熟练掌握文件的各种读写操作函数。
3. 掌握文件定位的方法。

二、实训准备

1. 复习文件的打开、关闭函数。
2. 复习文件的各种读写操作函数。

三、实训内容

实训题目1（9-1.c）

输入10个学生的数学、政治和英语3门功课的成绩，统计各科的平均分后，将所有数据存入文件student.dat中，并实现输入一个学生的学号，输出该学生的平均成绩的功能。

【分析与提示】

定义一个包含10个元素的结构体类型数组，来存储10个学生的信息；在输入成绩的同时累加各科总成绩和每个学生的总成绩，所有学生的成绩录入完成后即可求出各科的平均成绩和每个学生的平均成绩；运用匹配方法在数组中查找是否有相同学号的学生，若有则输出该数组元素对应的平均成绩。

参考代码：

```
#include <stdio.h>
struct student
{
    int no;
    char name[8];
    int score[3];
    float ave;
}stu[10];
void main()
{
    int i,j,sum;
    int num;
    FILE *fp;
```

```
    for(i=0;i<10;i++)
    {
        printf("\ninput score of student %d\n", i+1);
        printf ("no.:");
        scanf ("%d", &stu [i] .no);
        printf ("name:");
        scanf ("%s", stu [i] .name);
        sum=0;
        for (j=0; j<3; j++)
         {
            printf ("score %d:", j+1);
            scanf ("%d", &stu [i] .score [j] );
            sum=sum+ stu [i] .score [j];
        }
        stu [i] .ave=sum/3.0;
    }
    fp=fopen ("student","w");
    for (i=0; i<10; i++)
    if (fwrite (&stu [i], sizeof (struct student), 1, fp)! =1)
    printf ("file write error!");
    fclose (fp);
    fp=fopen ("student","r");
    for (i=0; i<10; i++)
    fread (&stu [i], sizeof (struct student), 1, fp);
    printf ("input a no.:");
    scanf ("%d", &num);
    for (i=0; i<10; i++)
    if (num==stu [i] .no)
     {
        printf ("\n%d,%f", stu [i] .no, stu [i] .ave);
        break;
    }
}
```

实训题目 2（9-2.c）

将 10 名职工的数据从键盘输入，然后送到磁盘文件 worker 中保存，设职工数据包括工号、职工名、性别、年龄、工资，再从磁盘调入这些数据，依次打印出来（用 fread（）和 fwrite（）函数）。

【分析与提示】

运用 scanf（）函数键入职工信息，再运用 fwrite（）函数把数据保存到指定文件中，

运用 fread（）函数从文件中调取数据存储在数组中即可。

参考代码：

```
#include <stdio. h>
structstu
{
    char num[10];
    char name[8];
    char sex;
    int age;
    floatscore;
    }st[10];
    void main( )
    {
        int i;
        FILE *fp;
        for(i=0;i<10;i++)
        {
            printf("\n请输入职工信息%d\n", i+1);
            printf ("工号:");
            scanf ("%s", st [i] .num);
            printf ("职工名:");
            scanf ("%s", st [i] .name);
            printf ("性别:");
            scanf ("%c", &st [i] .sex);
            printf ("年龄:");
            scanf ("%d", &st [i] .age);
            printf ("工资:");
            scanf ("%f", &st [i] .score);
        }
    fp=fopen ("worker","w");
    for (i=0; i<10; i++)
    if (fwrite (&st [i], sizeof (structstu), 1, fp)! =1)
    printf ("file write error!");
    fclose (fp);
    fp=fopen ("worker","r");
    for (i=0; i<10; i++)
     {
        fread (&st [i], sizeof (struct st), 1, fp);
        printf ("%s,%s,%c,%d,%f\ \n", st [i] .num, st [i] .name, st [i] .sex, st [i]
```

```
.age, st [i] .score);
        }
    }
```

四、实训报告要求

将以上各题的源程序、运行结果，在实训中遇到的问题和解决问题的方法，以及实训心得体会填写在实训报告上。

第二部分　C 语言程序设计习题

1. 认识 C 语言

一、填空题

1. 一个 C 语言函数由_____和_____两部分组成。
2. 在 C 语言中，源程序文件的后缀名是_____。
3. 一个源程序不论由多少个文件组成，都有一个且只能有一个_____。
4. 源程序经过编译后产生的结果称为_____，其扩展名为_____。
5. 一个 C 程序总是从_____开始执行的。

二、选择题

1. 下面选项中，不能用作 C 语言标识符的是（　　）。

A. print　　B. FOR　　C. &a　　D. _00

2. 在 C 语言中，源程序文件的后缀名是（　　）。

A. c　　B. obj　　C. exe　　D. bas

3. C 语言可执行程序的开始执行点是（　　）。

A. 程序中的一条可执行语句　　B. 程序中第一个函数

C. 程序中的 main（）函数　　D. 包含文件中的第一个函数

4. 以下叙述中不正确的是（　　）。

A. 一个 C 源程序必须包含一个 main（）函数

B. 一个 C 源程序可由一个或多个函数组成

C. C 程序的基本组成单位是函数

D. 在 C 程序中，注释说明只能位于一条语句的后面

5. 以下叙述中正确的是（　　）。

A. 在 C 程序中，“｛”“｝”可以不成对出现

B. 在 C 程序中，Aph 和 aph 代表不同的标识符

C. 在 C 程序中，注释部分可以用两个“/＊”括起来

D. 在 C 程序中，符号常量和变量一样的使用

三、简答题

1. 概述 C 语言的主要特点。

2. C 语言程序的运行一般要经过哪几个步骤?

四、在 Visual C++ 6.0 环境下运行程序，观察运行结果

1. 输入以下程序并运行。

```
# include <stdio. h>
void main( )
{
    int a1,a2,x;
    a1=50;
    a2=100;
    x=a1+a2;
    printf("x=%d\n", x);
}
```

2. 输入以下程序并运行。

```
# include <stdio. h>
void main( )
{
    int a1,a2,x;
    a1=20;
    a2=30;
    x=a1*a2;
    printf("x=%d\n", x);
}
```

3. 输入以下程序并运行。

```
# include <stdio. h>
void main( )
{
    printf("☆☆☆☆☆☆\n");
    printf ("☆          ☆\n");
    printf ("☆          ☆\n");
    printf ("☆          ☆\n");
    printf ("☆          ☆\n");
    printf ("☆☆☆☆☆☆\n");
}
```

五、编程题

1. 编写一个简单的 C 程序，用于显示以下信息：

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

Welcome to GuangDong NanFang institute of Technolngy!

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

2. 编写一个简单的 C 程序，从键盘输入两个整数，求它们的差并输出。

2. 数 据 类 型

一、选择题

1. 下面字符常量正确的是（　　）。

A. "c"　　B. ' \ \ '　　C. ' n'　　D. ' ab'

2. 下列符号串中，属于 C 语言合法标识符的个数为（　　）。

_1_2_3,a-b-c,float,9cd,a3b4

A. 1　　B. 2　　C. 3　　D. 4

3. 以下全部都是合法的用户标识符的是（　　）。

A. user　PAD#d　stu-age　　B. scnafa10　_345　A * stu

C. age　extern　xbb　　D. stu_name　NameChar　_1day

4. 假设所有变量均为整型，则表达式（a=2，b=5，b++，a+b）的值是（　　）。

A. 7　　B. 8　　C. 6　　D. 2

5. 下列转义字符不正确的是（　　）。

A. ' \ \ '　　B. ' \ '　　C. ' \ 053'　　D. ' \ 0'

6. 下列选项中属于不正确的赋值语句的是（　　）。

A. t++;　　B. n1=（n2=（n3=0））;

C. k=i==j;　　D. a=3=b;

7. 设“int x=2，y=1;”，则表达式（! x| | y--）的值是（　　）。

A. -1　　B. 0　　C. 1　　D. 2

8. 下列标识符不是关键字的是（　　）。

A. break　　B. char　　C. return　　D. swithch

9. C 语言中运算对象必须是整型的运算符是（　　）。

A. %　　B. /　　C. !　　D. *

10. 当 c 的值不为 0 时，在下列选项中能正确将 c 的值赋给变量 a、b 的是（　　）。

A. c=b=a;　　B.（a=c）| |（b=c);

C.（a=c）&&（b=c);　　D. a=c=b;

11. 以下选项中可作为 C 语言合法常量的是（　　）。

A. -80　　B. -080　　C. -8e1.0　　D. -80.0e

12. 下列不属于字符型常量的是（　　）。

A. ' A'　　B. ' \ 117'　　C. "a"　　D. ' \ x93'

13. 若 x、b、m、n 均为 int 型变量，执行下面语句后 b 的值为（　　）。

```
m=20;n=6;
x=(--m==++n)?--m:++n;
b=m++;
```

A. 11　　B. 6　　C. 19　　D. 18

14. 设 a=5，b=6，c=7，d=8，m=2，n=2，执行（m=a>b）&&（n=c>d）后 m，n 的值为（　　）。

A. 2，2　　B. 2，0　　C. 0，2　　D. 0，0

15. 若已定义 x 和 y 为 double 类型，则表达式“x=1，y=x+3/2”的值是（　　）。

A. 2.0　　B. 2　　C. 1　　D. 2.5

16. 下列选项中，优先级最高的运算符是（　　）。

A. &&　　B. *=　　C. !=　　D. []

17. 若运行时给变量 x 输入 12，则以下程序的运行结果是（　　）。

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

A. 0　　B. 22　　C. 12　　D. 10

18. 已知各变量的类型说明如下：

```
int k,a,b;
unsigned w=5;
double x=1.42;
```

则以下不符合 C 语言语法的表达式是（　　）。

A. x%（-3）　　B. w+=-2

C. k=（a=2，b=3）　　D. a+=a-=a=3

19. 执行下列程序中的输出语句后，x 的值是（　　）。

```
#include <stdio.h>
void main()
{
    int x;
    printf("%d\n", (x=5*6, x*2, x+20));
}
```

A. 30　　B. 60　　C. 50　　D. 80

20. 下列程序的输出结果是（　　）。

```
#include <stdio.h>
main()
{
    int x=1,y=0,z;
    z=(x<=0)&&(y-->=0);
    printf("%d %d %d\n", z, x, y);
}
```

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

21. 字符串"y=5*4\n"的长度为（　　）。

A. 6　　B. 7　　C. 8　　D. 9

22. 设变量a是整型，b是实型，c是双精度型，则表达式2+'a'+b*c值的数据类型是（　　）。

A. int　　B. float　　C. double　　D. char

23. 设x，y均为int型变量，则以下不合法的赋值语句是（　　）。

A. x*y=x+y;　　B. y=（x%2）/10;

C. x*=y+8;　　D. x=y=0;

24. 设有变量说明：int x = 5，y = 3；那么表达式y=x > y ?（x = 1）:（y = -1）运算后，x和y的值分别是（　　）。

A. 1和-1　　B. 1和1　　C. 5和-1　　D. 1和3

二、填空题

1. 表达式是由运算符和_____串接起来所组成的符号序列。

2. 若a=1，b=2，c=3，d=4，判断下列各式结果：

（1）a>=b&&a<=b

结果是_____。

（2）a<b||b!=a

结果是_____。

（3）a>=c||b>=d

结果是_____。

（4）b>=c&&c!=d

结果是_____。

3. C语言程序中的数据可以分为_____和_____两大类。其中，_____是指在程序执行过程中值不改变的量。_____是程序中用于存储信息的单元。

4. 请写出下列表达式的值。

（1）3.0*2+2*7-'A'

结果是_____。

（2）29/3+34%3+3.5

结果是_____。

（3）45/2+（int）3.14159/2

结果是_____。

（4）a=3*5，a=b=3*2

结果是_____。

5. 程序填空

任意输入一个有五位数字的正整数 x，分别输出每一数位上的数字（由高到低分别用 b5，b4，b3，b2，b1 表示），请将程序补充完整。

```
#include <stdio.h>
main()
{
    int b1,b2,b3,b4,b5;
    _____ x;
    scanf("%ld", &x);
    b5=_____;
    b4= (x/1000)%10;
    b3=_____;
    b2= (x/10)%10;
    b1=_____;
    printf ("the number is_____ \n", x);
    printf ("its bit is:%d,%d,%d,%d,%d\n", b5, b4, b3, b2, b1);
}
```

6. 程序改错（注意：题中的编号为行序号，并非程序本身）

计算任意一个半径为 *r* 的圆的面积和周长（结果保留两位小数）

```
(1)#include <stdio.h>
(2)main()
{
(3)float r; p=3.14,c,area;
(4)printf("input r:");
(5)scanf("%f",r);
(6)c=2p*r;
(7)area=p*r*r
(8)printf("c=%-7.2f,area=%-7.2f",c,area);
(9)}
```

错误语句的行号：（ ）

改正后的语句全行：_____;

错误语句的行号：（ ）

改正后的语句全行：_____；

错误语句的行号：（　）

改正后的语句全行：_____；

错误语句的行号：（　）

改正后的语句全行：_____。

3. 控 制 结 构

一、选择题

1. 判断 char 型变量 ch 是否为大写字母的正确表达式是（　　）。

A. 'A'<=ch<='Z'　　B.（ch>='A'）&&（ch<='Z'）

C.（ch>='A'）&（ch<='Z'）　　D.（'A'<= ch）||（'Z'>= ch）

2. 为了避免在嵌套的条件语句 if…else 中产生二义性，C 语言规定，else 子句是与（　　）配对。

A. 缩排位置相同的 if　　B. 其之前最近的 if

C. 其之后最近的 if　　D. 同一行上的 if

3. 若有程序段如下：

```
a=b=c=0; x=35;
if(! a)
x--;
else if(b)
;
if(c)
x=3;
else
x=4;
```

执行后，变量 x 的值是（　　）。

A. 34　　B. 3　　C. 35　　D. 4

4. 有 switch 语句如下：

```
switch(k)
{
    case 1:s1; break;
    case 2:s2; break;
    case 3:s3; break;
    default:s4;
}
```

与它的功能相同的程序段是（　）。

A. if（k = 1）s1;
　if（k = 2）s2;
　if（k = 3）s3;
　else s4;

B. if（k == 1）s1; if（k == 2）s2;
　if（k == 3）s3;
　else s4;

C. if（k == 1）s1; break;
　if（k == 2）s2; break;
　if（k == 3）s3; break;
　else s4;

D. if（k == 1）s1;
　if（k == 2）s2;
　if（k == 3）s3;
　if（!（（k == 1）||（k == 2）||（k == 3））） s4;

5. 若有程序如下，执行后，打印结果是（　）。

```
main()
{int a,b,c;
    a=1;b=2;c=3;
    if(a>b)
    if(b<0)
    c=0;
    else
    c+=1;
    printf("%d\n", c);
}
```

A. 3　　B. 4　　C. 2　　D. 1

6. 以下描述不正确的是（　　）。

A. 使用 while 和 do…while 循环时，循环变量初始化的操作应在循环体语句之前完成

B. while 循环是先判断表达式，后执行循环语句

C. do…while 和 for 循环均是先执行循环语句，后判断表达式

D. for、while 和 do…while 循环中的循环体均可以由空语句构成

二、填空题

1. 若变量 x、y、z 都是 int 型的。现有语句：“scanf（"%d,%d,%d", &x, &y, &z);”，为了使 x 值是 20，y 值是 14，z 值是 28，应该在键盘上键入_____。

2. while、do…while 和 for 循环结构至少执行一次循环的是_____。

3. 循环：“for（x=0; x ! = 123;　）scanf（"%d", &x);”在输入 x 等于_____

时被终止。

4. break 语句只能用于_____语句和_____语句。

5. 在循环控制中，break 语句用于结束_____，continue 语句用于结束_____。

6. 设“int x = 10;”，则循环语句“while （x > = 1） x - -;”执行后，x 的值是_____。

7. 下面程序的功能是输出以下形式的金字塔图案，请填空：

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

```
#include <stdio. h>
void main( )
{
    int i,j;
    for( i=1;i<=4;i++)
    {
        for(j=1;j<=4-i;j++)printf(" ");
        for （j=1; j<=_____; j++)
        printf （" * "）;
        printf （" \ n "）;
    }
}
```

三、程序分析题

1. 写出以下程序的运行结果。

```
#include<stdio. h>
main( )
{
    int x=1,total=0,y;
    while(x<=5)
    {
        y = x * x;
        total += y;
        ++x;
    }
    printf(“\nTotal is %d\n”,total);
}
```

2. 写出以下程序的运行结果。

```
#include <stdio. h>
void main( )
{     int i,j,n=0,m=0;
```

```
    for(j = 0; j < 10; j++)
    {
        if((j%2)&&(j%3))
        m++;
        else
        n++;
    }
    printf("m=%d n=%d\n", m, n);
}
```

3. 写出以下程序的运行结果。

```
#include <stdio.h>
void main()
{
    int i,j;
    for(i=j=1;j<=50;j++)
    {
        if(i>=10)break;
        if(i%2)
    {
    i+=5;
    continue;
}
i-=3;
}
printf("j=%d\n", j);
}
```

4. 阅读下面的程序：

```
#include<stdio.h>
void main()
{int x;
    scanf("%d", &x);
    if (x>=4)
    while (x--);
    printf ("%d\n", ++x);
}
```

如果输入为 5，则其输出是什么？

5. 写出以下程序的运行结果。

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

```
    for(i=0;i<=5;i++)
    {   i=i*2;
        printf("%d",i);
    }
}
```

四、编程题

1. 设圆半径 $r=1.5$，圆柱高 $h=3$，求圆周长、圆面积、圆球表面积、圆球体积、圆柱体积。请编程序，用 scanf（）输入数据，输出计算结果，输出时要求文字说明，取小数点后两位数字。

2. 编写一个程序，从键盘接收一个算术运算符和两个整数。根据运算符的不同，求出相应的算术运算结果，打印输出。

3. 利用 while、do…while、for 循环语句，分别编写程序求 1+2+3+…+99+100 之和，并打印输出。

4. 求 2！+4！+…+10！的和。

5. 给一个百分制成绩，要求输出等级“A”“B”“C”“D”“E”。90 分以上为“A”，80~90 分为“B”，70~79 分为“C”，60 分以下为“D”。

6. 输入 4 个整数，要求按由大到小的顺序输出。

7. 输入两个正整数 m 和 n，求其最大公约数和最小公倍数。

8. 输入一行字符，分别统计出其中英文字母、空格、数字和其他字符的个数。

9. 一球从 100 米高度自由下落，每次落地后返回原高度的一半，再落下。求它在第 10 次落地时共经过多少米？第 10 次反弹多高？

10. 打印以下图案：

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

4. 函　　数

一、选择题

1. 若函数的定义为：

```
fun(char ch)
```

```
{
    … …
}
```

那么该函数的返回值是（　　）。

A. void 型　　B. char 型　　C. float 型　　D. int 型

2. 阅读下面的程序，给出执行后全局变量 gx 的取值（　　）。

```
#include "stdio.h"
int gx;
void sgb()
{
    int gx;
    gx = 3;
}
void fun()
{
    gx = 5;
    sgb();
    gx = gx * 3;
}
main()
{
    fun();
    printf("gx=%d\n",gx);
}
```

A. 15　　B. 0　　C. 9　　D. 5

3. 下列程序的运行结果是（　　）。

```
fun(int a,int b)
{    if(a<b)
    return(a);
    else
    return(b);
}
main()
{ int x=3,y=8,z=1,r=7;
    r=fun(fun(x,y),2*z);
    printf("%d\n",r);
}
```

A. 8　　B. 7　　C. 2　　D. 3

4. C 语言规定：简单变量作实参时，它和对应形参之间的数据传递方式是（　　）。

A. 地址传递

B. 单向值传递

C. 由实参传给形参，再由形参传回给实参

D. 由用户指定的传递方式

5. 有以下程序：

```
void f(int x,int y)
{ int t;
    if(x<y){ t=x; x=y; y=t; }
}
main()
{ int a=4,b=3,c=5;
    f(a,b); f(a,c); f(b,c);
    printf("%d,%d,%d\n", a, b, c);
}
```

执行后输出的结果是（　　）。

A. 3，4，5　　B. 5，3，4　　C. 5，4，3　　D. 4，3，5

6. C 语言中函数返回值的类型是由（　　）决定的。

A. 调用该函数的主调函数类型

B. return 语句中的表达式类型

C. 定义函数时所指定的返回函数值类型

D. 调用函数时临时

7. 下面函数定义中正确的是（　　）。

A. double fun（double f1. double f2）{　}

B. double fun（double f1；double f2）{　}

C. double lun（double f1，double f2）；{　}

D. double fun（double f1，f2）{　}

8. 下面跳转语句中，可以选择不唯一的跳转目的地的是（　　）。

A. continue；　　B. break；　　C. goto 标识符；　　D. return；

9. 在 C 程序中，若对函数类型未加说明，则函数的隐含类型为（　　）。

A. int　　B. double　　C. void　　D. ch

10. C 语言规定，在一个源程序中 main（）函数的位置（　　）。

A. 必须在程序的最前面　　B. 必须在程序的最后面

C. 必须在预处理命令的后面　　D. 可以在其他函数之前或之后

11. 下面关于函数的叙述，正确的是（　　）。

A. 在函数体中可以直接引用另一个函数中声明为 static 类别的局部变量的值

B. 在函数体中可以调用函数自身

C. 在函数体中可以定义另一个函数

D. 在函数体中至少必须有一个 return

12. 在定义任何一个函数时，下列选项中不可缺少的是（　　）。

A. 函数名前的数据类型　　B. 函数名后的一对圆括号

C. 形参声明　　D. 函数体中的语句

13. 关于函数返回值，下面叙述中正确的是（　　）。

A. 函数返回值的类型由函数体内 return 语句包含的表达式的类型决定

B. 若函数体内没有 return 语句，则函数没有返回值

C. 若函数体中有多个 return 语句，则函数的返回值是排列在最后面的 return 语句中表达式的值

D. 函数返回值的类型由函数头部定义的函数类型决定

14. 已有函数 fun 的定义“void fun（void）｛printf（"That's great!"）;｝”，则调用 fun（）函数的正确形式是（　　）。

A. fun;　　B. fun（）;　　C. fun（void）;　　D. fun（1）;

15. 若已定义一个有返回值的函数，则下面关于调用该函数的叙述中错误的是（　　）。

A. 函数调用可以作为一个函数形参　　B. 函数调用可以出现在表达式中

C. 函数调用可以作为一个函数实参　　D. 函数调用可以作为独立的语句存在

16. 已有函数 fun 的定义“int fun（int a，int b）｛（if（a< b）return（a，b）; else return（b，a）;）”，在 main（）函数中若调用函数（3，4），得到的返回值是（　　）。

A. 3　　B. 4　　C. 3 和 4　　D. 4 和 3

二、填空题

1. C 程序总是从 _____函数开始执行。

2. C 语言中，在函数调用时使用的参数称为_____；在函数定义时，函数头中列出的参数称为 _____。

3. 如果一个函数没有返回值，那么该函数的类型是_____。

4. 一个函数在它的函数体内调用它自身称为_____。

5. 一个函数的形式参数的作用域是_____。

三、编程题

1. 补充下列程序中的函数 bt（）。函数的功能：根据每个职工的工资统计出某单位在发工资时，共需要多少张 100 元、50 元、10 元、1 元的人民币并统计出工资总额。已知主函数如下：

```
int a=0,b=0,c=0,d=0;
void main()
{
    float x1,sum=0;
    printf("输入一个职工工资=");
    scanf ("%f", &x1);
```

```
    while（x1！=-1.0）
     {
        bt（x1）;
        sum=sum+x1;
        printf（"输入一个职工工资="）;
        scanf（"%f", &x1）;
     }
     printf（"工资总额为%.2f其中：\n", sum）;
     printf（"100元共计%d张\n 50元共计%d张\n 10元共计%d张\n 1元共计%d张 \n", a, b, c, d）;
  }
```

2. 编写函数，输出 x 的 n 次幂。

3. 用递归函数求 m 和 n 的最大公约数。

4. 编写一个求前 n 个自然数二次方和的函数 squ（），n 由调用者传递过来。用函数 main（）加以验证。

5. 数　　组

一、选择题

1. 在 C 语言中，引用数组元素时，其数组下标的数据类型允许的是（　　）。

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

2. 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］

3. 若有说明：int a［3］［4］；则对 a 数组元素的正确引用是（　　）。

A. a［2］［4］　　B. a［1，3］　　C. a［1+1］［0］　　D. a（2）（1）

4. 设有数组定义：char array［ ］="China"；则 strlen（array）的值为（　　）。

A. 4　　B. 5　　C. 6　　D. 7

5. 设有数组定义：char array［ ］="China"；则数组 array 所占的存储空间为（　　）。

A. 4 个字节　　B. 5 个字节　　C. 6 个字节　　D. 7 个字节

二、填空题

1. 以下程序的运行结果是_____。

```
main()
{
    int i,k,a[10],p[3];
    k=5;
    for(i=0;i<10;i++)
    a[i]=i;
```

```
    for(i=0;i<3;i++)
    p[i]=a[i*(i+1)];
    for(i=0;i<3;i++)
    k=k+p[i]*2;
    printf("%d\n", k);
}
```

2. 以下程序的运行结果是_____。

```
main()
{
    int k,a[6]={1,2,3,4,5,6};
    for(k=5;k>0;--k)
    if(a[k]%2==0)
    printf("%d", a[k]);
}
```

3. 以下程序的运行结果是_____。

```
main()
{
    char ch[7]={"65ab21"};
    int i, s=0;
    for (i=0; ch[i]>='0'&&ch[i]<='9'; i+=2)
    s=10*s+ch[i]-'0';
    printf ("%d\n", s);
}
```

4. 以下程序的运行结果是_____。

```
main()
{
    char a[]="Monday", b[]="day";
    strcpy (a, b);
    printf ("%s\t%s\n", a, b);
    printf ("%c\t%c\n", a[4], a[5]);
}
```

5. 以下程序的运行结果是_____。

```
main()
{
    char a[80]="AB", b[80]="LMNP";
    int i=0;
    strcat (a, b);
    while (a[i++]!='\0')
    b[i]=a[i];
```

```
    puts (b);
}
```

三、编程题

1. 用一维数组，求 10 个数的平均值。

2. 从键盘任意输入 10 个整数，求偶数的个数。

3. 输入 10 个整数，用一维数组保存，统计其中正数、负数和零的个数，并在屏幕上输出。

4. 用数组编程，求 Fibonacci 数列的前 20 个数据（1，1，2，3，5，8，…）。

5. 用数组输入 5 个学生成绩，求出这些成绩的平均值，并输出所有高出平均值的分数。

6. 有 10 个数据（45，52，65，74，76，79，81，90，92，100），请任意输入一个数，检查是否与 10 个数的其中一个相等。若相等，请输出该数和该数的位置。若不相等，输出“没有找到”。

7. 从键盘上输入 10 个整数，并放入一个一维数组中，然后将其前 5 个元素与后 5 个元素对换，即：第 1 个元素和第 10 个元素互换，第 2 个元素与第 9 个元素互换……分别输出数组原来的值和对换后各元素的值。

8. 将两个二维数组对应元素加起来，存到另一个二维数组中。

```
  10  20        1  4
a=30  40    b=2  5
  50  60        3  6
```

9. 在给定的数组中找出与输入数据一样的数据，并给出所在的行号和列号。

10. 定义一个 6×5（6 行 5 列）的二维数组，求每一行的最小值。

11. 计算并显示 3 个学生信息，求三人数学、英语、计算机考试的总分和平均分。

12. 打印以下的杨辉三角形（要求打印出 10 行），格式如下：

请输入杨辉三角的行数 10

```
1
1    1
1    2    1
1    3    3    1
1    4    6    4    1
1    5   10   10    5    1
1    6   15   20   15    6    1
1    7   21   35   35   21    7    1
1    8   28   56   70   56   28    8    1
1    9   36   84  126  126   84   36    9    1
```

13. 请把一个字符串的中间某部分字符截取出来，例如：you are student！把 are 截取出来。

14. 编写程序，统计一个字符串中 26 个大小写字母出现的次数。

15. 数字中有一个很有趣的现象，任何一个 4 位整数 n，只要它的 4 位数字不完全相同，把它的 4 位数字从大到小排序生成一个数 p，再把它的 4 位数字从小到大排序生成另外一个数 q，计算 p 与 q 的差 $r=p-q$，那么 r 可能等于 6 174。如果 r 不是 6 174，则再把 r 看成 4 位数（不足高位补 0），重复同样的操作，最终 r 必等于 6 174，编写程序验证这一现象。

6. 指　　针

一、简答题

1. 指针是指什么？

2. 有以下定义：int a［4］ = ｛0，1，2，3｝，＊p；

若“p=&a［1］;”，则＊p++的值是多少？若“p=&a［2］;”，则＊--p 的值是多少？

3. 在函数调用中，用指针变量作为函数的参数进行传递的好处是什么？

二、运行程序写结果

1. 写出以下程序的运行结果。

```
#include <stdio.h>
void main()
{
    int x[8]={8,7,6,5,0,0},*s;
    s=x+3;
    printf("%d",s[2]);
}
```

2. 写出以下程序的运行结果。

```
#include <stdio.h>
void main()
{
    int a=7,b=8,*p,*q,*r;
    p=&a;q=&b;
    r=p;p=q;q=r;
    printf("%d,%d,%d,%d",*p,*q,a,b);
}
```

3. 写出以下程序的运行结果。

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

```
void main( )
{
    char a[ ] = "language" , b [ ] = "programe" ;
    char * p, * q;
    p=a; q=b;
    while ( * p&& * q)
     {
        if ( ( * p) = = ( * q) ) printf ("%c", * p);
        p++; q++;
    }
}
```

4. 写出下面程序的运行结果。

```
#include <stdio. h>
void fun(char * c,int d)
{
    * c= * c+1;d=d+1;
    printf("%c,%c,", * c, d);
}
void main ( )
{
    char b='a' , a='A' ;
    fun (&b, a); printf ("%c,%c\ \n", b, a);
}
```

5. 写出下面程序的运行结果。

```
#include <stdio. h>
void swap(int * a,int * b)
{
    int * t;
    t=a;a=b;b=t;
}
void main( )
{
    int x=3,y=5, * p=&x, * q=&y;
    swap(p,q);
    printf("%3d%3d\ \n", * p, * q);
}
```

6. 写出下面程序运行的结果。

```
#include <stdio. h>
void swap1(int x,int y)
```

```
{
    int t;
    t=x;x=y;y=t;
    return;
}
void swap2(int *x,int *y)
{
    int t;
    t=*x;*x=*y;*y=t;
    return;
}
void main()
{
    int x=3,y=5;
    printf("%d,%d\n", x, y);
    swap1 (x, y);
    printf ("%d,%d\n", x, y);
    swap2 (&x, &y);
    printf ("%d,%d\n", x, y);
}
```

三、编程题

1. 实现字符串原样复制。
2. 统计一个字符串中的单词个数。
3. 用指针实现合并两个字符串。
4. 从输入的三个字符串中找出最长的一个字符串并输出。
5. 编写用指针删除字符串中空格的函数。
6. 编写用指针合并两个字符串的函数。

7. 结构体与共用体

一、简答题

1. 结构体类型与以前的标准数据类型有什么区别?
2. 结构体类型与共用体类型有什么异同?
3. 枚举类型适用于什么场合?
4. 类型定义有什么意义?

二、编程题

1. 编写一程序，定义一个点的坐标，然后定义两个点，求这两个点之间的距离。

2. 请编写程序：将下表所示的数赋给结构体数组并按照年龄从小到大的顺序将它们输出到屏幕上。

姓名	年龄	年薪/元
Zhangsan	38	58 000
Lisi	22	32 000
WangWu	24	37 000

3. 假设有三个学生，每个学生的数据包括学号、姓名及三科成绩。要求从键盘输入各学生的数据，输出三科的总平均成绩及最高分学生的情况。

4. 跳水比赛评分程序。

在跳水比赛中运动员每完成一个跳水动作，都有八名裁判对其打分，但每次动作的总成绩中都要去掉一个最高分和一个最低分。每个运动员需要完成十个动作，每次动作结束都要按当前的总成绩为全部运动员重新排序，结果在运动员出场比赛时显示出来。

设总共有三名运动员参加比赛，其编号分别为：1、2、3。

8. 文　　件

一、填空题

1. C 文件按编码方式分为_____和 ASCII 文件。

2. 在 C 语言中，称指向 FILE 型结构变量的指针为_____。

3. 通过文件指针就可对它所指的文件进行打开、关闭、读、写等各种操作，FILE *p 把变量 p 说明为一个文件指针。这里用到的“FILE”，是在_____头文件里定义的。

4. 打开一个已存在的二进制文件，只能读取数据，在文件打开模式_____。

二、程序阅读题

1. 阅读下面的程序，说明程序的功能。

```
#include<stdio.h>
main()
{
    FILE *fp;
    char ch,fname[30];
    printf("Enter file name:");
    gets(fname);
    if((fp=fopen(fname,"w"))==NULL)
    {
        printf("File could not be opened! \n");
```

```
        exit(0);
    }
    while((ch=getchar())! = '#')
    fputc(ch,fp);
    fclose(fp);
}
```

2. 阅读下面的程序，说明程序的功能。

```
#include<stdio. h>
struct stu
{
    char name[10];
    int num;
    int age;
    char addr[15];
}boya[2],boyb[2], * pp, * qq;
main()
{
    FILE * fp;
    char ch;
    int i;
    pp=boya;
    qq=boyb;
    if((fp=fopen("stu_list","wb+")) ==NULL)
     {
        printf ("Cannot open file strike any key exit!");
        getch ();
        exit (1);
    }
    printf ("\ninput data\ n");
    for (i=0; i<2; i++, pp++)
    scanf ("%s%d%d%s", pp->name, &pp->num, &pp->age, pp->addr);
    pp=boya;
    fwrite (pp, sizeof (struct stu), 2, fp);
    rewind (fp);
    fread (qq, sizeof (struct stu), 2, fp);
    printf ("\ n\ nname\ tnumber        age        addr\ n");
    for (i=0; i<2; i++, qq++)
    printf ("%s\ t%5d%7d     %s\ n", qq->name, qq->num, qq->age, qq->addr);
    fclose (fp);
}
```

第三部分　综合项目开发

综合项目　学生通讯录

【功能描述】

编写一个菜单驱动的通讯录，其功能要求如下：

1. 打印：能打印全部学生的信息；
2. 查找：能按照多种方式查找某一位同学的通讯信息；
3. 更改：能更改某些同学的通讯信息；
4. 删除：能删除某一位同学的通讯信息；
5. 插入：能够插入某位同学的通讯信息；
6. 排序：能够按学号对通讯录信息进行排序；
7. 退出：结束程序。

【系统设计】

本系统主要模块有 6 个：打印、查找、更改、删除、插入、排序。

系统的整体设计如图 3-1 所示：

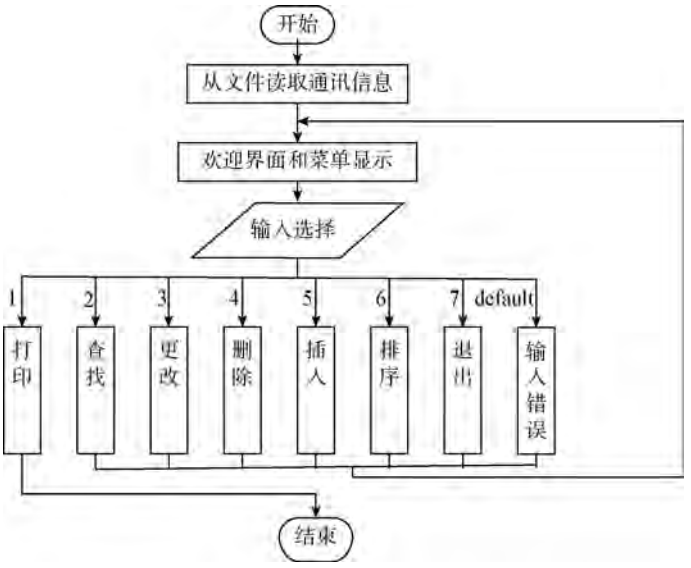


图 3-1　系统整体设计图

下面分别对 6 个模块进行介绍。

1. 打印

该模块主要用来显示所有学生的通讯信息。

2. 查找

该模块能够根据用户选择不同信息资料进行查找，其流程图如图 3-2 所示。

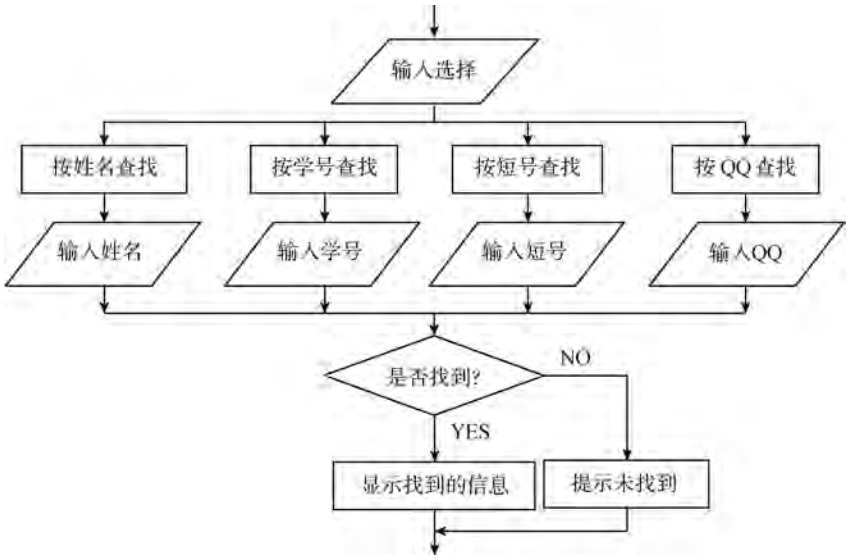


图 3-2 查找模块流程图

3. 更改

该模块能够根据需要修改学生的信息，通过学号查询到要修改的学生，若找到该学号对应的学生则对其进行相关的修改，否则提示可以增加该学生的信息。最后显示修改后的信息。其流程图如图 3-3 所示。

图 3-3 更改模块流程图

4. 删除

该模块主要用于删除指定学生的通讯信息。要删除某个学生信息，首先要查询到该学生，然后才能够进行删除。若查找不到，给出提示。

5. 插入

先输入要插入的人数，然后进行逐个插入。

6. 排序

该模块主要将所有学生的通讯信息按照学号从小到大的顺序来排列显示。

【关键技术】

1. 使用结构体数组存放数据

使用结构体数组存放数据，先定义长度足够多的数组（本例事先定义了长度为 100 的结构体数组），存放学生的通讯录数据。

在本项目中，结构体类型可定义为：

```
struct addresslist
{
    int num;//学号
    char name[20];//姓名
    char sex[10];//性别
    char place[20];//籍贯
    char phone[20];//短号
    char QQ[12];//QQ
}student[100];
```

使用结构体数组，因为事先已经定义了数组的大小，所以不必像链表那样动态地分配内存空间。

2. 程序设计思路

本项目关键在于查找、删除、修改、插入、排序等操作。

查找：遍历结构体数组的每一个元素，找出符合条件的信息并输出。

删除：删除结构体数组一个或多个元素，主要使用数组元素前移的方式进行删除。

插入：在结构体数组的最后增加一个或多个元素。

修改：遍历结构体数组的每一个元素，找出符合条件的信息，重新输入该数组元素的相关信息。

3. 文件的读写

本项目首先要从文件中读取通讯录文件，对文件的删除、插入、修改均写入文件。

4. 函数的灵活使用

本项目包含多项功能，各项功能都有自己独立的函数，并用结构体指针进行参数传递，增强了函数的可移植性和重用性。

本项目功能点的划分及函数定义见表 3-1。

表 3-1 项目功能点划分及函数定义

功能点	函数编号	函数及功能简述
整体功能	1	main（）
菜单显示	2	int menu（）
插入	3	void insert（）/*添加多条记录，调用函数 8，10*/
查看	4	void print（struct addresslist *p）/*显示所有记录，调用函数 10*/
查询	5	void find（struct addresslist *p）/*查询记录，调用函数 10*/
更改	6	void change（struct addresslist *p）;/*更改学生通讯信息，调用函数 5，8，10*/
删除	7	void dele（struct addresslist *p）;/*删除一条记录，调用函数 5，8，10*/
保存	8	void save（）/*保存更改后或删除后的信息 */
排序	9	void sort（struct addresslist *p）;/*排序，调用函数 8，10，4*/
返回主菜单	10	char returnToMenu（void）/*返回主菜单*/
退出		void end（）

【程序实现】

```
#include "stdio.h"
#include "stdlib.h"
#include "string.h"
int studentNum=0; /*全局变量，记录学生人数*/
int flag=0; /*全局变量，记录执行删除和插入后人数的变化*/
struct addresslist
{
    int num;
    char name[20];
    char sex[10];
    char place[20];
    char phone[20];
    char QQ[12];
} student[100];
/*以下是函数原型声明*/
int menu(); /*菜单界面*/
void end(); /*退出系统*/
void save(); /*保存通讯录信息到文件中*/
void print(struct addresslist *p); /*输出某位同学的通讯录信息*/
void input(struct addresslist *p, int n); /*录入某位同学的通讯录信息*/
```

```
void sort (struct addresslist *p); /*按照学号大小排序*/
void find (struct addresslist *p); /*查找某位同学的通讯录信息*/
void dele (struct addresslist *p); /*删除某位同学的通讯录信息*/
void insert (); /*插入某位同学的通讯录信息*/
void change (struct addresslist *p); /*更改某位同学的通讯录信息*/
char returnToMenu (void); /*返回主菜单*/
void main ()
{
    int i, peopleNum;
    FILE *fp;
    char chose;
    if ( (fp=fopen ("通讯录.txt","r+")) ==NULL) {
        printf ("文件打不开! \n");
        exit (0);
    }
    i=0;
    do {
        i=0;
        while (! feof (fp)) {
            fread (&student [i], sizeof (struct addresslist), 1, fp);
            i++;
        studentNum++;
    }
    switch (menu ()) {
        case 1:
            system ("cls");
            print (student);
            chose=returnToMenu ();
            break;
        case 2:
            find (student-flag);
            chose=returnToMenu ();
            break;
        case 3:
            system ("cls");
            change (student);
            chose=returnToMenu ();
            break;
        case 4:
```

```
            dele (student);
            chose=returnToMenu ();
            break;
        case 5:
            system ("cls");
            printf ("请输入要录入的人数\n");
            scanf ("%d", &peopleNum);
            while (peopleNum--) {
                insert ();
            }
            chose=returnToMenu ();
            break;
        case 6: system ("cls");
            sort (student);
            chose=returnToMenu ();
            break;
        case 7:
            system ("cls");
            end ();
            break;
        default:
            printf ("输入错误");
            chose=returnToMenu ();
            break;
        }
    } while (chose!='Y'&&chose!='y');
    if (chose=='Y'||chose=='y') {
        system ("cls");
        end ();
    }
    fclose (fp);
}
/*菜单显示*/
int menu ()
{
    int num;
    system ("cls");
    printf ("**************************通讯录**************************\n\n");
    printf ("\t1. 打印    2. 查找    3. 更改\n\n");
```

```
        printf ("\t4. 删除        5. 插入        6. 排序\n\n");
        printf ("\t7. 退出\n\n");
        printf ("*************************************************************\n");
        printf ("请输入你想要进行的操作数:");
        fflush (stdin);
        scanf ("%d", &num);
        return num;
    }
    /*将更改后的数据写入文件中保存*/
    void save ()
    {
        int i;
        FILE *fp;
        if ( (fp=fopen ("通讯录.txt","w") ) ==NULL) {
        printf ("\n\t\t文件打开失败");
        }
        for (i=0; i<studentNum+flag-1; i++) {
            if (fwrite (&student [i], sizeof (addresslist), 1, fp)! =1) {
                printf ("\n\t\t写入文件错误! \n");
            }
        }
        fclose (fp);
        printf ("\n\t\t通讯录文件已保存\n");
    }
    /*在屏幕上输出通讯录信息*/
    void print (struct addresslist *p)
    {
        int i;
        for (i=studentNum+flag-1; i>0; i--, p++)
         {
            printf ("学号%d\n姓名:%s\t\t性别:%s\n籍贯:%s\n短号:%s\nQQ:%s", p->
num, p->name, p->sex, p->place, p->phone, p->QQ);
            printf ("\n*************************************************************\n");
        }
    }
    /*录入某个同学的通讯信息*/
    void input (struct addresslist *p, int n)
    {
        static int  i=0;
            printf ("请输入第%d个同学的资料: \n", ++i);
            printf ("请输入学号:");
            scanf ("%d", &p->num);
```

```
    printf ("请输入姓名:");
    scanf ("%s", p->name);
    printf ("性别:");
    scanf ("%s", p->sex);
    printf ("请输入籍贯:");
    scanf ("%s", p->place);
    printf ("短号:");
    scanf ("%s", p->phone);
    printf ("QQ:");
    scanf ("%s", p->QQ);
    printf ("\n");
    printf ("**********录入完成!!**********\n");
}
/*按学号从小到大的顺序对通讯录信息进行排序*/
void sort (struct addresslist *p)
 {
    int i, j;
    struct addresslist temp;
    printf ("\t\t\t按学号排序后:\n");
    for (i=0; i<studentNum+flag-1; i++)
        for (j=i+1; j<studentNum+flag-1; j++)
        if ( (p[i].num) >= (p[j].num) )
         {
            temp=p[i];
            p[i] =p[j];
            p[j] =temp;
        }
    print (student);
}
/*按照多种方式查找某一位同学的通讯信息*/
void find (struct addresslist *p)
 {
    int n, numb, i, z=6;
    char name2[20], cellnum[10], QQ1[11];
    system ("cls");
    printf ("请选择查找方式:\n");
    printf ("\t\t\t1. 按姓名查找\n");
    printf ("\t\t\t2. 按学号查找\n");
    printf ("\t\t\t3. 按短号查找\n");
```

```
            printf ("\t\t\t4. 按 QQ 查找\n");
            scanf ("%d", &n);
            switch (n) {
                case 1:
                    printf ("请输入要查找的姓名:");
                    fflush (stdin);
                    scanf ("%s", name2);
                    for (i=0; i<studentNum; p++, i++)
                    if (strcmp ( (p->name), name2) ==0)
                     {
                        printf ("\n\n%s 同学的资料如下:\n\n", name2);
                        printf ("学号%d\n姓名:%s\n性别:%s\n籍贯:%s\n短号:%s\nQQ:%
s", p->num, p->name, p->sex, p->place, p->phone, p->QQ);
                        break;
                    }
                    if (i==studentNum)
                    printf ("查无此人\n");
                    break;
                case 2:
                printf ("\n请输入你要查询同学的学号:");
                scanf ("%d", &numb);
                for (i=0; i<studentNum; p++, i++)
                    if ( (p->num) ==numb) {
                    printf ("学号为%d 的同学的资料如下:\n", numb);
                    printf ("学号%d\n姓名:%s\t\t性别:%s\n籍贯:%s\n短号:%s\nQQ:%
s", p->num, p->name, p->sex, p->place, p->phone, p->QQ);
                    break;
                }
                if (i==studentNum)
                printf ("查无此人\n");
                break;
            case 3:
                printf ("请输入要查找短号:");
                fflush (stdin);
                scanf ("%s", cellnum);
                for (i=0; i<studentNum+flag; p++, i++)
                if (strcmp ( (p->phone), cellnum) ==0) {
                printf ("\n\n短号%s 的同学的资料如下:\n\n", cellnum);
                printf ("学号%d\n姓名:%s\n性别:%s\n籍贯:%s\n短号:%s\nQQ:%s", p->
```

```
num, p->name, p->sex, p->place, p->phone, p->QQ);
                    break;
                }
                if (i==studentNum)
                printf ("查无此人\n");
                break;
                case 4:
                    printf ("请输入要查找 QQ:");
                    fflush (stdin);
                    scanf ("%s", QQ1);
                    for (i=0; i<studentNum+flag; p++, i++)
                    if (strcmp ( (p->QQ), QQ1) ==0) {
                    printf ("\n\nQQ 为%s 同学的资料如下: \n\n", QQ1);
                    printf ("学号%d\n 姓名:%s\n 性别:%s\n 籍贯:%s\n 短号:%s\nQQ:%s", p->
num, p->name, p->sex, p->place, p->phone, p->QQ);
                    break;
                }
                if (i==studentNum)
                printf ("查无此人\n");
                break;
                    default:
                    printf ("输入错误\n");
                }
        }
        /*删除某一位同学的通讯信息*/
        void dele (struct addresslist *p)
         {
            int num, i=0, j;
            char choice;
            system ("cls");
            printf ("请输入所要删除同学的学号:");
            scanf ("%d", &num);
            for (i=0; i<=studentNum+flag; p++, i++)
                if ( (p->num) ==num) {
                    printf ("\n 学号为%d 的同学的资料如下: \n", num);
                    printf ("学号%d\n 姓名:%s\t 性别:%s\n 籍贯:%s\n 短号:%s\nQQ:%
s", p->num, p->name, p->sex, p->place, p->phone, p->QQ);
                    printf ("\n 是否确定删除 Y/N");
                    fflush (stdin);
```

```
                choice=getchar();
                if(choice=='Y'||choice=='y'){
                for(j=i-1;j<=studentNum+flag;j++)
                student[j+1]=student[j+2];//
                printf("\t已经成功删除…\n");
                flag--;
                save();
            }
            else
                printf("已经放弃删除…\n\n");
        }
        if(i==studentNum+flag)
            printf("查无此人\n");
    }
    /*插入某位同学的通讯信息*/
    void insert()
     {
        printf("请输入你要插入同学的资料:\n");
        input(&student[studentNum+flag-1],1);
        flag++;
        save();
    }
    /*更改某些同学的通讯信息*/
    void change(struct addresslist *p)
     {
        int i;
        int num;
        int enter;
        char choice;
        printf("请输入你更改资料同学的学号");
        scanf("%d",&num);
        for(i=0;i<studentNum;p++,i++)
        if((p->num)==num){
        printf("学号为%d的同学的资料如下:\n",num);
        printf("*************************************************************\n");
        printf("学号:%d\n姓名:%s\t\t性别:%s\n籍贯:%s\n短号:%s\nQQ:%s",p->
num,p->name,p->sex,p->place,p->phone,p->QQ);
        printf("*************************************************************\n");
        printf("\n[1]更改该同学学号;\n[2]更改该同学姓名;\n[3]更改该同学性
别;\n[4]更改该同学籍贯;\n[5]更改该同学短号;\n[6]更改该同学QQ;\n");
        do{
```

```
printf ("请选择你要更改的选项: \n");
scanf ("%d", &enter);
while (enter>6 || enter<1)
 {
    printf ("输入错误!! \n 请再次选择.... \n");
    getchar ();
    scanf ("%d", &enter);
}
switch (enter) {
    case 1:
        printf ("请输入该同学的新学号: \n");
        scanf ("%d", &p->num);
        break;
    case 2:
        printf ("请输入该同学的姓名: \n");
        scanf ("%s", p->name);
        break;
    case 3:
        printf ("请输入该同学的性别: \n");
        scanf ("%s", p->sex);
        break;
    case 4:
        printf ("请输入该同学的籍贯: \n");
        scanf ("%s", p->place);
        break;
    case 5:
        printf ("请输入该同学的短号: \n");
        scanf ("%s", p->phone);
        break;
    case 6:
        printf ("请输入该同学的 QQ: \n");
        scanf ("%s", p->QQ);
        break;
    }
    printf ("\n 还需要更改其他的么? 需要的话请按 Y/y_ _ ");
    getchar ();
    scanf ("%c", &choice);
} while (choice=='y' || choice=='Y');
save ();
printf ("更改后新资料为: \n");
printf ("学号%d\n 姓名:%s\t\t 性别:%s\n 籍贯:%s\n 短号:%s\nQQ:%s",
```

```
p->num, p->name, p->sex, p->place, p->phone, p->QQ);
                    break;
                }
                /*若找不到更改的学生，询问是否添加*/
                if (i==studentNum) {
                    printf ("查无此人\n是否添加 y/n_");
                    char add;
                    getchar ();
                    scanf ("%c", &add);
                    if (add=='y' || add=='Y') {
                        insert ();
                    }
                }
        }
    /*返回菜单界面*/
    char returnToMenu (void)
    {
        char ch;
        printf ("\n退出请按 Y，否则请按其他键_   ");
        fflush (stdin);
        ch=getchar ();
        return (ch);
    }
    /*退出系统*/
    void  end ()
    {
        printf ("**************************谢谢使用 ******************************\n");
        exit (0);
    }
```

【运行结果】

1. 欢迎界面和菜单显示

欢迎界面如图 3-4 所示。

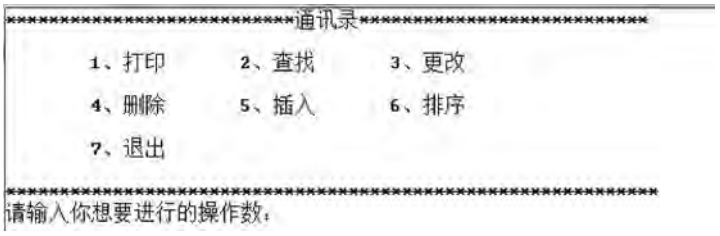


图 3-4　欢迎界面

2. 打印

选择 1 进入，按回车键之后结果如图 3-5 所示：

图 3-5 打印界面

3. 查找

选择 2 进入，结果如图 3-6 所示：

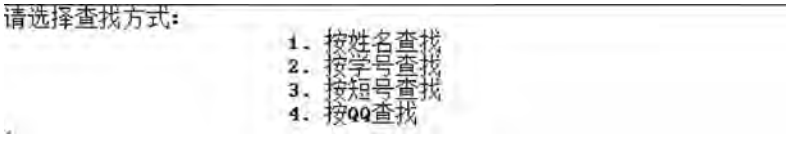


图 3-6 查找界面

输入 1~4 选择查找的方式。

4. 更改

回到主菜单，选择 3 进入。输入要更改同学的学号，再按 1~6 选择要更改的内容，结果如图 3-7 所示：

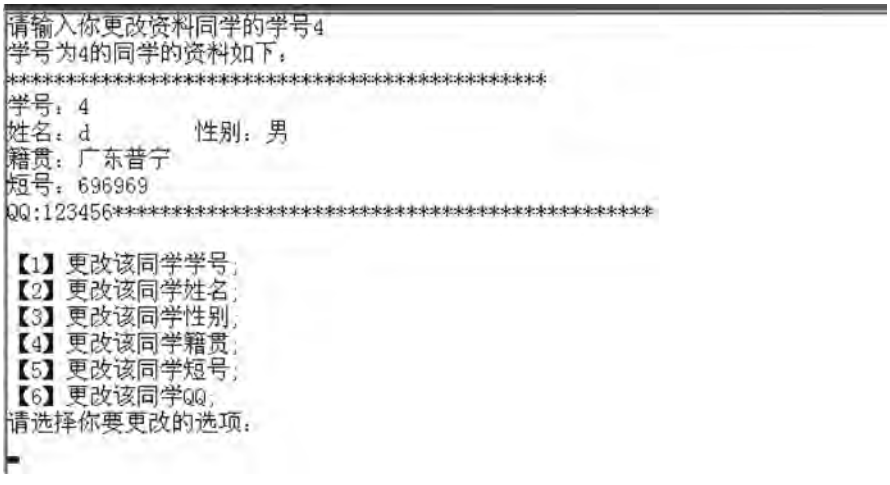


图 3-7 更改界面

5. 删除

回到主菜单，选择 4 进入。输入要删除的同学的学号，按回车键后，再按 Y 确认删除。结果如图 3-8 所示：

```
请输入所要删除同学的学号：1

学号为1的同学的资料如下：
学号1
姓名：令狐冲                性别：男
籍贯：广东
短号：696969
QQ:1234567
是否确定删除Y/N
```

图 3-8　删除界面

6. 插入

回到主菜单，选择 5 进入。输入要插入的同学个数，然后输入要插入的资料。结果如图 3-9 所示：

```
请输入要录入的人数
2
请输入你要插入同学的资料：
请输入第1个同学的资料：
请输入学号：4
请输入姓名：老张
性别：男
请输入籍贯：广东普宁
短号：692833
QQ：123456

**************************录入完成!!**************************

                通讯录文件已保存
请输入你要插入同学的资料：
请输入第2个同学的资料：
请输入学号：5
请输入姓名：啊杰
性别：男
请输入籍贯：广东普宁
短号：656565
QQ：654321

**************************录入完成!!**************************

                通讯录文件已保存

退出请按Y，否则请按其他键_
```

图 3-9　插入界面

7. 排序

选择 6 进入。回车之后结果如图 3-10 所示：

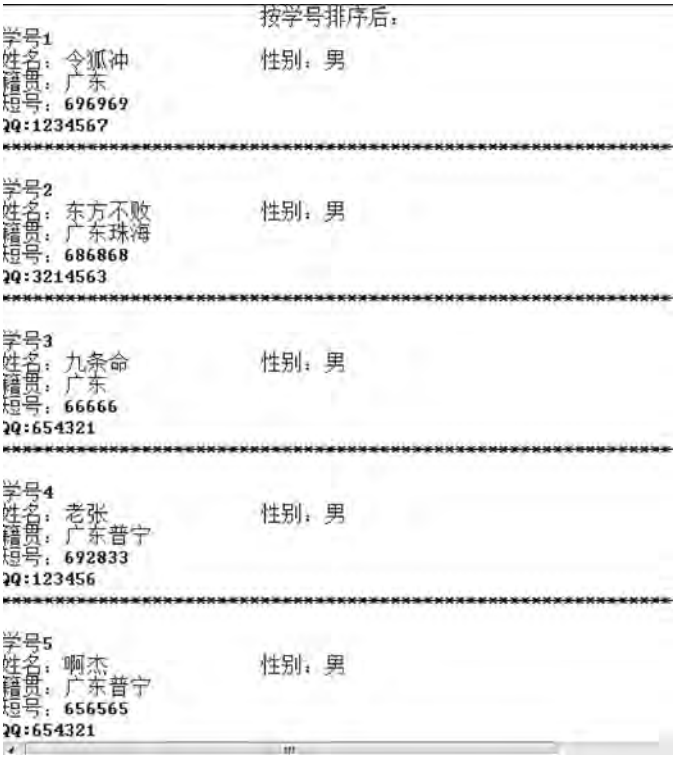


图 3-10 排序界面

第四部分　全国计算机等级考试二级考试指导

全国计算机等级考试二级　C 语言程序设计考试大纲（2018 年版）

基本要求

1. 熟悉 Visual C++集成开发环境。

2. 掌握结构化程序设计的方法，具有良好的程序设计风格。

3. 掌握程序设计中简单的数据结构和算法并能阅读简单的程序。

4. 在 Visual C++集成环境下，能够编写简单的 C 程序，并具有基本的纠错和调试程序的能力。

考试内容

一、C 语言程序的结构

1. 程序的构成，main 函数和其他函数。

2. 头文件、数据说明、函数的开始和结束标志以及程序中的注释。

3. 源程序的书写格式。

4. C 语言的风格。

二、数据类型及其运算

1. C 语言的数据类型（基本类型、构造类型、指针类型、无值类型）及其定义方法。

2. C 语言运算符的种类、运算优先级和结合性。

3. 不同类型数据间的转换与运算。

4. C 语言表达式类型（赋值表达式、算术表达式、关系表达式、逻辑表达式、条件表达式、逗号表达式）和求值规则。

三、基本语句

1. 表达式语句、空语句、复合语句。

2. 输入/输出函数的调用，正确输入数据并正确设计输出格式。

四、选择结构程序设计

1. 用 if 语句实现选择结构。

2. 用 switch 语句实现多分支选择结构。

3. 选择结构的嵌套。

五、循环结构程序设计

1. for 循环结构。

2. while 和 do…while 循环结构。

3. continue 语句和 break 语句。

4. 循环的嵌套。

六、数组的定义和引用

1. 一维数组和二维数组的定义、初始化和数组元素的引用。

2. 字符串与字符数组。

七、函数

1. 库函数的正确调用。

2. 函数的定义方法。

3. 函数的类型和返回值。

4. 形式参数与实际参数，参数值的传递。

5. 函数的正确调用，嵌套调用，递归调用。

6. 局部变量和全局变量。

7. 变量的存储类别（自动、静态、寄存器、外部），变量的作用域和生存期。

八、编译预处理

1. 宏定义和调用（不带参数的宏、带参数的宏）。

2. “文件包含”处理。

九、指针

1. 地址与指针变量的概念，地址运算符与间址运算符。

2. 一维、二维数组和字符串的地址以及指向变量、数组、字符串、函数、结构体的指针变量的定义。通过指针引用以上各类型数据。

3. 用指针作函数参数。

4. 返回地址值的函数。

5. 指针数组，指向指针的指针。

十、结构体（即“结构”）与共用体（即“联合”）

1. 用 typedef 说明一个新类型。

2. 结构体和共用体类型数据的定义和成员的引用。

3. 通过结构体构成链表，单向链表的建立，节点数据的输出、删除与插入。

十一、位运算

1. 位运算符的含义和使用。

2. 简单的位运算。

十二、文件操作

只要求缓冲文件系统（即高级磁盘 I/O 系统），对非标准缓冲文件系统（即低级磁盘 I/O 系统）不要求。

1. 文件类型指针（FILE 类型指针）。

2. 文件的打开与关闭（fopen，fclose）。

3. 文件的读写（fputs，fgets，fread，fwrite，fprintf，fscanf 函数的应用），文件的定位（rewind，fseek 函数的应用）。

考试方式

上机考试，考试时长 120 分钟，满分 100 分。

一、题型及分值

单项选择题 40 分（含公共基础知识部分 10 分）。

操作题 60 分（包括程序填空题、程序修改题及程序设计题）。

二、考试环境

操作系统：中文版 WINDOWS 7。

开发环境：Microsoft Visual C++ 2010 学习版。

全国计算机二级考试公共基础知识考点

第一章　数据结构与算法

考点 1　算法的基本概念

算法是指对解决问题过程的准确完整的描述，算法也就是解决问题的操作步骤。

（1）算法的基本特征：可行性、确定性、有穷性、拥有足够的情报。

（2）算法的基本要素。

一个算法由两种基本要素组成：一是对数据对象的运算和操作；二是算法的控制结构。

1）算法中对数据的运算和操作。

在一般的计算机系统中，基本的运算和操作有以下四类：算术运算、逻辑运算、关系运算和数据传输。

2）算法的控制结构：算法中各操作之间的执行顺序称为算法的控制结构。

描述算法的工具通常有传统流程图、N-S 结构化流程图、算法描述语言等。一个算法一般都可以用顺序、选择、循环三种基本控制结构组合而成。

考点 2 算法复杂度

算法的复杂度用来衡量算法的优劣，包括算法的时间复杂度和算法的空间复杂度。

1. 算法的时间复杂度

算法的时间复杂度是指执行算法所需要的计算工作量。

同一个算法用不同的语言实现，或者用不同的编译程序进行编译，或者在不同的计算机上运行，效率均不同。这表明使用绝对的时间单位衡量算法的效率是不合适的。撇开这些与计算机硬件、软件有关的因素，可以认为算法的计算工作量是用算法所执行的基本运算的次数来衡量的，依赖于问题的规模（通常用整数 n 表示），它是问题规模的函数。即

$$算法的工作量=f(n)$$

2. 算法的空间复杂度

算法的空间复杂度是指执行这个算法所需要的内存空间。

一个算法所占用的存储空间包括算法程序所占的空间、输入的初始数据所占的存储空间以及算法执行过程中所需要的额外空间。其中额外空间包括算法程序执行过程中的工作单元以及某种数据结构所需要的附加存储空间。如果额外空间量相对于问题规模来说是常数，则称该算法是原地工作的。在许多实际问题中，为了减少算法所占的存储空间，通常采用压缩存储技术，以便尽量减少不必要的额外空间。

考点 3 数据结构的定义

数据结构作为计算机的一门学科，主要研究和讨论以下三个方面内容：

（1）数据集合中各数据元素之间所固有的逻辑关系，即数据的逻辑结构。

（2）在对数据元素进行处理时，各数据元素在计算机中的存储关系，即数据的存储结构。

（3）对各种数据结构进行的运算。

数据是对客观事物的符号表示，在计算机科学中是指所有能输入到计算机中并被计算机程序处理的符号的总称。

数据元素是数据的基本单位，在计算机程序中通常作为一个整体进行考虑和处理。

数据对象是性质相同的数据元素的集合，是数据的一个子集。

数据的逻辑结构是对数据元素之间的逻辑关系的描述，它可以用一个数据元素的集合和定义在此集合中的若干关系来表示。数据的逻辑结构有两个要素：一是数据元素的集合，通常记为 D；二是集合上的关系，它反映了数据元素之间的前后件关系，通常记为 R。一个数据结构可以表示成：

$$B=(D, R)$$

其中 B 表示数据结构，D 是数据元素的集合，R 是 D 上关系的集合，反映了 D 和数据元素之间的前后件关系，用二元组来表示。

除了可以用二元组来表示，数据结构还可以用图形来表示，引出三个基本概念：

（1）根节点：数据结构中没有前件的节点。

（2）叶子节点：数据结构中没有后件的节点。

（3）内部节点：数据结构中除了根节点和叶子节点以外的节点。

数据的逻辑结构在计算机存储空间中的存放形式称为数据的存储结构（也称数据的物理结构）。

由于数据元素在计算机存储空间中的位置关系可能与逻辑关系不同，因此，为了表示存放在计算机存储空间中的各数据元素之间的逻辑关系（即前后件关系），在数据的存储结构中，不仅要存放各数据元素的信息，还需要存放各数据元素之间的前后件关系的信息。

一种数据的逻辑结构根据需要可以表示成多种存储结构，常用的存储结构有顺序、链接、索引等存储结构。而采用不同的存储结构，其数据处理的效率是不同的。因此，在进行数据处理时，选择合适的存储结构是很重要的。

考点 4　线性结构与非线性结构

根据数据结构中各数据元素之间前后件关系的复杂程度，一般将数据结构分为两大类型：线性结构与非线性结构。如果一个非空的数据结构满足下列两个条件：有且只有一个根节点；每一个节点最多有一个前件，也最多有一个后件。那么，该数据结构为线性结构。线性结构又称线性表。在一个线性结构中插入或删除任何一个节点后还应是线性结构。如果一个数据结构不是线性结构，则称之为非线性结构。

线性表要么是空表，要么可以表示为（a_1，a_2，…，a_n），其中 a_i（$i=1$，2，…，n）是线性表的元素，也称为线性表的节点。

通常线性表采用顺序存储和链接存储两种存储结构。

考点 5　栈和队列

1. 栈的基本概念

栈是特殊的线性表，是限定只在一端进行插入与删除的线性表，通常称插入、删除的一端为栈顶，另一端为栈底。当表中没有元素时称为空栈。栈顶元素总是最后被插入的元素，从而也是最先被删除的元素；栈底元素总是最先被插入的元素，从而也是最后才能被删除的元素。栈的修改原则是“先进后出”或“后进先出”。

用一维数组 S（1：m）作为栈的顺序存储空间，其中 m 为最大容量。

在栈的顺序存储空间 S（1：m）中，S（bottom）为栈底元素，S（top）为栈顶元素。top=0 表示栈空；top=m 表示栈满。

栈的基本运算有三种：入栈、退栈与读栈顶元素。

（1）入栈运算：入栈运算是指在栈顶位置插入一个新元素。首先将栈顶指针加一（即 top 加 1），然后将新元素插入到栈顶指针指向的位置。当栈顶指针已经指向存储空间的最后一个位置时，说明栈空间已满，不可能再进行入栈操作。这种情况称为栈“上溢”错误。

（2）退栈运算：退栈是指取出栈顶元素并赋给一个指定的变量。首先将栈顶元素

（栈顶指针指向的元素）赋给一个指定的变量，然后将栈顶指针减一（即 top 减 1）。当栈顶指针为 0 时，说明栈空，不可进行退栈操作。这种情况称为栈的“下溢”错误。

（3）读栈顶元素：读栈顶元素是指将栈顶元素赋给一个指定的变量。这个运算不删除栈顶元素，只是将它赋给一个变量，因此栈顶指针不会改变。当栈顶指针为 0 时，说明栈空，读不到栈顶元素。

2. 队列及基本运算

（1）队列的定义。队列是一种特殊的线性表，只允许在表的头部（front 处）进行删除操作，在表的尾部（rear 处）进行插入操作。进行插入操作的一端称为队尾，进行删除操作的一端称为队头。队列是“先进先出”或“后进后出”的线性表。

（2）队列的运算。队列是一种先进先出（First In First Out，FIFO）的线性表。它只允许在表的一端（队尾/rear）插入元素，而在另一端（队头/front）删除元素。插入操作称为入队或进队，删除操作称为出队或离队。队列示意如图 4-1：

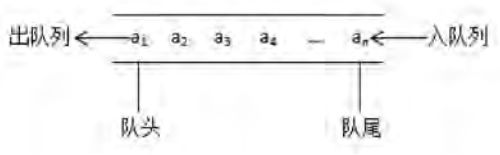


图 4-1 队列的运算

1）顺序队列：队列的顺序存储结构需要使用一个数组和两个整型变量来实现，数组用于存储队列中的所有元素，两个整型变量分别用于存储队头元素和队尾元素的下标位置，分别称为队头指针和队尾指针。

2）循环队列：将数组的前端和后端连接起来，形成循环的顺序表。

考点 6 线性链表

在链式存储方式中，要求每个节点由两部分组成：一部分用于存放数据元素值，称为数据域，另一部分用于存放指针，称为指针域。其中指针用于指向该节点的前一个或后一个节点（即前件或后件）。

链式存储方式既可用于表示线性结构，也可用于表示非线性结构。

（1）线性链表。

线性表的链式存储结构称为线性链表。在某些应用中，对线性链表中的每个节点设置两个指针，一个称为左指针，用以指向其前件节点；另一个称为右指针，用以指向其后件节点。这样的表称为双向链表。

（2）带链的栈。

栈也是线性表，也可以采用链式存储结构。带链的栈可以用来收集计算机存储空间中所有空闲的存储节点，这种带链的栈称为可利用栈。

考点7　树与二叉树

满二叉树也是完全二叉树，而完全二叉树一般不是满二叉树。应该注意二者的区别。

1. 树的基本概念

树（Tree）是一种简单的非线性结构，由一个或多个节点组成的有限集合 T，其中有一个特定的称为根的节点，其余节点可分为 m（$m \geq 0$）个互不相交的有限集 T_1，T_2，T_3，…，T_m，每一个集合本身又是一棵树，且称为根的子树。

节点（Node）：树中的元素，包含数据项及若干指向其子树的分支。

节点的度（Degree）：节点拥有的子树数。

节点的层次：从根节点开始算起，根为第一层。

叶子（Leaf）：度为零的节点，也称端节点。

孩子（Child）：节点子树的根称为该节点的孩子节点。

双亲（Parent）：孩子节点的上层节点，称为这些节点的双亲。

兄弟（Sibling）：同一双亲的孩子。

深度（Depth）：树中节点的最大层次数。

森林（Forest）：M 棵互不相交的树的集合。

树的存储结构可以采用具有多个指针域的多重链表，节点中指针域的个数应由树的度来决定。在实际应用中，这种存储结构并不方便，一般将树转化为二叉树表示。

2. 二叉树及其基本性质

（1）二叉树的定义。二叉树是一种很有用的非线性结构，具有以下两个特点：

1）非空二叉树只有一个根节点；

2）每一个节点最多有两棵子树，且分别称为该节点的左子树和右子树。

二叉树是一种重要的树形结构，其结构定义为：二叉树是 n（$n \geq 0$）个节点的有限集，它或为空树（$n=0$），或由一个根节点和两棵分别称为根的左子树和右子树的、互不相交的二叉树组成。

（2）二叉树的基本性质。二叉树具有以下几个性质：

1）在二叉树的第 k 层上，最多有 2^k-1（$k \geq 1$）个节点；

2）深度为 m 的二叉树最多有 2^m-1 个节点；

3）在任意一棵二叉树中，度为 0 的节点（即叶子节点）总是比度为 2 的节点多一个。

4）具有 n 个节点的二叉树，其深度至少为 $[\log_2 n]+1$，其中 $[\log_2 n]$ 表示取 $\log_2 n$ 的整数部分。

3. 满二叉树与完全二叉树

满二叉树：在一棵二叉树中，如果所有分支节点都存在左子树和右子树，而且所有叶子节点都在同一层上，在满二叉树的第 k 层上有 2^k-1 个节点，且深度为 m 的满二叉树有 2^m-1 个节点。

完全二叉树：如果一棵具有 n 个节点的二叉树的结构与满二叉树的前 n 个节点的结构相同，称为完全二叉树。满二叉树是特殊的完全二叉树。

对于完全二叉树来说，叶子节点只可能在层次最大的两层上出现：对于任何一个节

点，若其右分支下的子孙节点的最大层次为 p，则其左分支下的子孙节点的最大层次或为 p，或为 $p+1$。

完全二叉树具有以下两个性质：

1）具有 n 个节点的完全二叉树的深度为 $[\log_2 n]+1$。

2）设完全二叉树共有 n 个节点。如果从根节点开始，按层次（每一层从左到右）用自然数 1，2，…，n 给节点进行编号，则对于编号为 k（$k=1$，2，…，n）的节点有以下结论：

若 $k=1$，则该节点为根节点，它没有父节点；若 $k>1$，则该节点的父节点编号为 INT（$k/2$）。

若 $2k \leqslant n$，则编号为 k 的节点的左子节点编号为 $2k$；否则该节点无左子节点（显然也没有右子节点）。

若 $2k+1 \leqslant n$，则编号为 k 的节点的右子节点编号为 $2k+1$；否则该节点无右子节点。

考点 8　二叉树的遍历

在遍历二叉树的过程中，一般先遍历左子树，再遍历右子树。在先左后右的原则下，根据访问根节点的次序，二叉树的遍历分为三类：前序遍历、中序遍历和后序遍历。

（1）前序遍历：先访问根节点，然后遍历左子树，最后遍历右子树；并且，在遍历左、右子树时，仍然先访问根节点，然后遍历左子树，最后遍历右子树。

（2）中序遍历：先遍历左子树，然后访问根节点，最后遍历右子树；并且，在遍历左、右子树时，仍然先遍历左子树，然后访问根节点，最后遍历右子树。

（3）后序遍历：先遍历左子树，然后遍历右子树，最后访问根节点；并且，在遍历左、右子树时，仍然先遍历左子树，然后遍历右子树，最后访问根节点。

考点 9　顺序查找

顺序查找的原理很简单，就是遍历整个列表，将逐个进行记录的关键字与给定值比较，若某个记录的关键字和给定值相等，则查找成功，找到所查的记录。如果直到最后一个记录，其关键字和给定值比较都不等时，则表中没有所查的记录，查找失败。

在下列两种情况下也只能采用顺序查找：

（1）如果线性表为无序表，则不管是顺序存储结构还是链式存储结构，只能用顺序查找。

（2）即使是有序线性表，如果采用链式存储结构，也只能用顺序查找。

考点 10　二分查找

二分查找的基本思想是，在有序表中，取中间记录作为比较对象，若给定值与中间记录的关键字相等，则查找成功；若给定值小于中间记录的关键字，则在中间记录的左半区继续查找；若给定值大于中间记录的关键字，则在中间记录的右半区继续查找。不断重复上述过程，直到找到为止。

时间复杂度为 O（$\log_2 N$）。

考点 11　排序技术

冒泡排序法和快速排序法都属于交换类排序法。

（1）冒泡排序法。首先，从表头开始往后扫描线性表，逐次比较相邻两个元素的大小，若前面的元素大于后面的元素，则将它们互换，不断地将两个相邻元素中的大者往后移动，最大者就到了线性表的最后。然后，从后到前扫描剩下的线性表，逐次比较相邻两个元素的大小，若后面的元素小于前面的元素，则将它们互换，不断地将两个相邻元素中的小者往前移动，最小者到了线性表的最前面。对剩下的线性表重复上述过程，直到剩下的线性表变空为止，此时已经排好序。

在最坏的情况下，冒泡排序需要比较次数为 $n（n-1）/2$。

（2）快速排序法。它的基本思想是，任取待排序序列中的某个元素作为基准（一般取第一个元素），通过一趟排序，将待排元素分为左右两个子序列，左子序列元素的排序码均小于或等于基准元素的排序码，右子序列的排序码则大于基准元素的排序码，然后分别对两个子序列继续进行排序，直至整个序列有序。

第二章　程序设计基础

重点学习知识点：

（1）结构化程序设计方法的四个原则。

（2）对象、类、消息、继承的概念，类与实例的区别。

考点 1　结构化程序设计

结构化程序设计方法的主要原则为自顶向下、逐步求精、模块化和限制使用 goto 语句。

结构化程序由三种基本结构组成：顺序结构、选择结构、循环结构。

考点 2　面向对象

当使用“对象”这个术语时，既可以指一个具体的对象，也可以泛指一般的对象，但是当使用“实例”这个术语时，必须是指一个具体的对象。

面向对象方法涵盖对象及对象属性、类、继承、多态性等几个基本要素。

（1）对象。

通常把对对象的操作也称为方法或服务。

属性即对象所包含的信息，它在设计对象时确定，一般只能通过执行对象的操作来改变。属性值指的是纯粹的数据值，而不能指对象。

操作描述了对象执行的功能，通过信息的传递，还可以为其他对象使用。

对象具有如下特征：标识唯一性、分类性、多态性、封装性、模块独立性。

（2）类。

类是具有共同属性、共同方法的对象的集合。它描述了属于该对象类型的所有对象的性质，而一个对象则是其对应类的一个实例。

类是关于对象性质的描述，它同对象一样，包括一组数据属性和在数据上的一组合法操作。

（3）消息。

消息是实例之间传递的信息，它请求对象执行某一处理或回答某一要求的信息，它统一了数据流和控制流。

一个消息由三部分组成：接收消息的对象的名称、消息标识符（消息名）和零个或多个参数。

（4）继承。

广义地说，继承是指能够直接获得已有的性质和特征，而不必重复定义它们。

继承分为单继承与多重继承。单继承是指一个类只允许有一个父类，即类等级为树形结构。多重继承是指一个类允许有多个父类。

（5）多态性。

对象根据所接收的消息而做出动作，同样的消息被不同的对象接收时可导致完全不同的行动，该现象称为多态性。

第三章 软件工程基础

经过对部分考生的调查以及对近年真题的总结分析，笔试部分经常考查的是软件生命周期、软件设计的基本原理、软件测试的目的以及软件调试的基本概念，对此部分应进行重点学习。

重点学习知识点：

（1）软件的概念、软件生命周期的概念及各阶段所包含的活动。

（2）概要设计与详细设计的概念、模块独立性及其度量的标准、详细设计常用的工具。

（3）软件测试的目的、软件测试的四个步骤。

（4）软件调试的任务。

考点1 软件定义与软件特点

软件指的是计算机系统中与硬件相互依存的另一部分，包括程序、数据和相关文档的完整集合。程序是软件开发人员根据用户需求开发的、用程序设计语言描述的、适合计算机执行的指令序列。数据是使程序能正常操纵信息的数据结构。文档是与程序的开发、维护和使用有关的图文资料。可见，软件由以下两部分组成：

（1）机器可执行的程序和数据；

（2）机器不可执行的，与软件开发、运行、维护、使用等有关的文档。

软件的特点如下：

（1）软件是逻辑实体，而不是物理实体，具有抽象性；

（2）没有明显的制作过程，可进行大量的复制；

（3）使用期间不存在磨损、老化问题；

（4）软件的开发、运行对计算机系统具有依赖性；

（5）软件复杂性高，成本昂贵；

（6）软件开发涉及诸多社会因素。

根据应用目标的不同，软件可分应用软件、系统软件和支撑软件（或工具软件）。

考点 2　软件工程过程与软件生命周期

考试链接：

考点 2 在笔试考试中出现的概率为 30%，主要是以选择题的形式出现，分值为 2 分，此考点为识记内容，读者应该识记软件生命周期的定义、主要活动阶段及其任务。

软件产品从提出、实现、使用、维护到停止使用的过程称为软件生命周期。一般包括可行性分析研究与需求分析、设计、实现、测试、交付使用以及维护等活动，如图 4-2 所示。

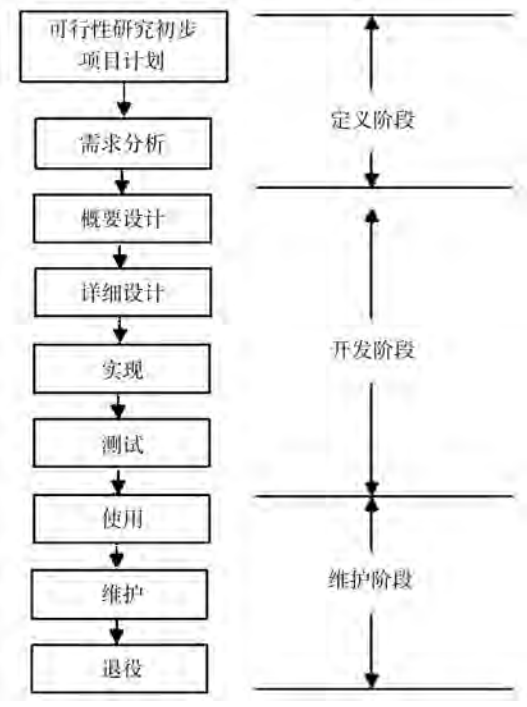


图 4-2　软件的生命周期

软件生命周期还可以分为软件定义、软件开发和软件运行维护三个阶段。

生命周期的主要活动是可行性研究与计划制定、需求分析、软件设计、软件实施、软件测试及运行与维护。

考点 3　软件设计及方法

软件设计应尽量做到高内聚、低耦合，即减弱模块之间的耦合性和提高模块内的内聚性，有利于提高模块的独立性。

1. 软件设计的基础

从技术观点上看，软件设计包括软件结构设计、数据设计、接口设计和过程设计。

（1）结构设计定义软件系统各主要部件之间的关系；

（2）数据设计将分析时创建的模型转化为数据结构的定义；

（3）接口设计是描述软件内部、软件和协作系统之间以及软件与人之间如何通信；

（4）过程设计则是把系统结构部件转换为软件的过程性描述。

从工程管理角度来看，软件设计分两步完成：概要设计和详细设计。概要设计将软件需求转化为软件体系结构、确定系统及接口、全局数据结构或数据库模式；详细设计确立每个模块的实现算法和局部数据结构，用适当方法表示算法和数据结构的细节。

2. 软件设计的基本原理

（1）抽象。软件设计中考虑模块化解决方案时，可以定出多个抽象级别。从概要设计到详细设计抽象的层次逐步降低。

（2）模块化。模块是指把一个待开发的软件分解成若干小的简单的部分。模块化是指解决一个复杂问题时自顶向下逐层把软件系统划分成若干模块的过程。

（3）信息隐蔽。信息隐蔽是指在一个模块内包含的信息（过程或数据），对于不需要这些信息的其他模块来说是不能访问的。

（4）模块独立性。模块独立性是指每个模块只完成系统要求的独立的子功能，并且与其他模块的联系最少且接口简单。模块的独立程度是评价设计好坏的重要度量标准。衡量软件的模块独立性使用耦合性和内聚性两个定性的度量标准。一般较优秀的软件设计，应尽量做到高内聚、低耦合，即减弱模块之间的耦合性和提高模块内的内聚性，有利于提高模块的独立性。

3. 概要设计

概要设计也称总体设计，其基本目标是能够针对软件需求分析中提出的一系列软件问题，概要地回答问题如何解决。

概要设计的任务：

（1）设计软件系统结构（简称软件结构）；

（2）数据结构及数据库设计；

（3）编写概要设计文档；

（4）评审。

4. 详细设计

详细设计的任务是为软件结构图中的每个模块确定实现的算法和局部数据结构，用某种选定的表达工具表示算法和数据结构的细节。

详细设计的常用工具如下：

（1）图形工具：程序流程图，N-S，PAD，HIPO。

（2）表格工具：判定表。

（3）语言工具：PDL（伪码）。

程序流程图的五种控制结构，即顺序型、选择型、先判断重复型、后判断重复型和多分支选择型。

方框图（N-S）含五种基本的控制结构，即顺序型、选择型、多分支选择型、WHILE重复型和UNTIL重复型。

PAD图表示五种基本控制结构，即顺序型、选择型、多分支选择型、WHILE重复型和UNTIL重复型。

过程设计语言（PDL）也称为结构化的语言和伪码，它是一种混合语言，采用英语的词汇和结构化程序设计语言，类似编程语言。

PDL可以由编程语言转换得到，也可以是专门为过程描述而设计的。

考点4　软件测试

软件测试是在软件投入运行前对软件需求、设计、编码的最后审核。其工作量、成本占总工作量、总成本的40%以上，而且具有较高的组织管理和技术难度。

（1）软件测试是为了发现错误而执行程序的过程；

（2）一个好的测试用例是能够发现至今尚未发现的错误的用例；

（3）一个成功的测试是发现了至今尚未发现的错误的测试。

测试的方法如下：

（1）静态测试和动态测试。

静态测试是指被测试程序不在机器上运行，而采用人工检测和计算机辅助静态分析的手段对程序进行检测。

动态测试是指在计算机上对运行程序进行软件测试。

（2）白盒测试和黑盒测试。

白盒就是透明的盒子，白盒测试是需要考虑内部结构的。

黑盒测试也称功能测试，黑盒就是一个黑色的盒子，主要用于集成测试、确认测试和系统测试中，一般包括等价类划分、边界值分析、判定表、因果图、状态图、随机测试、猜错法和正交试验法等。

考点6　软件测试的实施

软件测试过程分四个步骤，即单元测试、集成测试、验收测试和系统测试。

（1）单元测试。单元测试也称为模块测试。

（2）集成测试。集成测试的目的是检查模块之间，以及模块和已集成的软件之间的接口关系。

（3）验收测试。验收测试主要用于验证软件的功能、性能和其他特性是否与用户需求一致。根据用户的参与程度，通常包括以下几类：

Alpha测试：开发环境的测试；

Beta 测试：实际的环境下的测试；

验收测试：需要有甲方参与的测试。

（4）系统测试。系统测试主要是测试软件在真实系统环境下是否满足设计文档和合同的要求。系统测试的具体实施一般包括功能测试、性能测试、操作测试、配置测试、外部接口测试、安全性测试等。

考点 7　软件的调试

程序经调试改错后还应进行再测试，因为经调试后有可能产生新的错误，而且测试贯穿软件生命周期的整个过程。

在对程序进行了成功的测试之后将进入程序调试（通常称 Debug，即排错）。程序的调试任务是诊断和改正程序中的错误。调试主要在开发阶段进行。

程序调试活动由两部分组成，一是根据错误的迹象确定程序中错误的确切性质、原因和位置；二是对程序进行修改，排除这个错误。程序调试的基本步骤：

（1）错误定位。从错误的外部表现形式入手，研究有关部分的程序，确定程序中出错位置，找出错误的内在原因。

（2）修改设计和代码，以排除错误。

（3）进行回归测试（发现问题，修改后再次测试），防止引进新的错误。

调试原则可以从以下两个方面考虑：

（1）确定错误的性质和位置时的注意事项。分析思考与错误征兆有关的信息；避开死胡同；只把调试工具当作辅助手段来使用；避免用试探法，最多只能把它当作最后手段。

（2）修改错误原则。在出现错误的地方，很可能有别的错误。修改错误的一个常见失误是只修改了这个错误的征兆或这个错误的表现，而没有修改错误本身。注意修正一个错误的同时有可能会引入新的错误。修改错误的过程将迫使人们暂时回到程序设计阶段，修改源代码程序，不要改变目标代码。

第四章　数据库设计基础

重点学习知识点：

（1）数据的概念、数据库管理系统提供的数据语言、数据管理员的主要工作、数据库系统阶段的特点、数据的物理独立性及逻辑独立性、数据统一管理与控制、三级模式及两级映射的概念。

（2）数据模型三个描述内容、E-R 模型的概念及其 E-R 图表示法、关系操纵、关系模型三类数据约束。

（3）关系模型的基本操作、关系代数中的扩充运算。

（4）数据库设计生命周期法的四个阶段。

考点 1　数据、数据库、数据库管理系统

数据是数据库中存储的基本对象，描述事物的符号记录。

数据库是长期储存在计算机内、有组织的、可共享的大量数据的集合，它具有统一的结构形式并存放于统一的存储介质内，是多种应用数据的集成，并可被各个应用程序所共享。

数据库管理系统（Database Management System，DBMS）是数据库的机构，它是一种系统软件，负责数据库中的数据组织、数据操作、数据维护、数据控制及数据保护和数据服务等。数据库管理系统是数据系统的核心，主要有如下功能：数据模式定义、数据存取的物理构建、数据操纵、数据的完整性、安全性定义和检查、数据库的并发控制与故障恢复、数据的服务。

为完成数据库管理系统的功能，数据库管理系统提供相应的数据语言：数据定义语言、数据操纵语言、数据控制语言。

数据库管理员的主要工作包括数据库设计、数据库维护、改善系统性能和提高系统效率。

考点 2　数据库系统的发展

数据管理技术的发展经历了三个阶段，见表 4-1。

表 4-1　数据库管理技术发展阶段的特点

		人工管理阶段	文件系统阶段	数据库系统阶段
背景	应用背景	科学计算	科学计算、管理	大规模管理
	硬件背景	无直接存取存储设备	磁盘、磁鼓	大容量磁盘
	软件背景	没有操作系统	有文件系统	有数据库管理系统
	处理方式	批处理	联机实时处理、批处理	联机实时处理、分布处理、批处理
特点	数据的管理者	用户（程序员）	文件系统	数据库管理系统
	数据面向的对象	某一应用程序	某一应用	现实世界
	数据的共享程度	无共享，冗余度大	共享性差，冗余度大	共享性高，冗余度小
	数据的独立性	不独立，完全依赖于程序	独立性差	具有高度的物理独立性和一定的逻辑独立性
	数据结构化	无结构	记录内有结构、整体无结构	整体结构化，用数据模型描述
	数据控制能力	应用程序自己控制	应用程序自己控制	由数据库管理系统提供数据安全性、完整性、并发控制和恢复能力

考点 3 数据库系统的基本特点

数据独立性是数据与程序间的互不依赖性，即数据库中的数据独立于应用程序而不依赖于应用程序。

（1）数据结构化。

（2）数据的共享性高，冗余度低，易扩充。

（3）数据独立性高。

（4）数据由数据库管理系统统一管理和控制。

考点 4 数据库系统的内部结构体系

一个数据库只有一个概念模式。一个概念模式可以有若干个外模式。三级模式都有几种名称，读者应该熟记每个模式的另一些名称。

1. 数据库系统的三级模式

（1）概念模式，也称逻辑模式，是对数据库系统中全局数据逻辑结构的描述，是全体用户（应用）公共数据视图。一个数据库只有一个概念模式。

（2）外模式，外模式也称子模式，它是数据库用户能够看见和使用的局部数据的逻辑结构和特征的描述，它是由概念模式推导而来的，是数据库用户的数据视图，是与某一应用有关的数据的逻辑表示。一个概念模式可以有若干个外模式。

（3）内模式，内模式又称物理模式，它给出了数据库物理存储结构与物理存取方法。内模式处于最底层，它反映了数据在计算机物理结构中的实际存储形式，概念模式处于中间层，它反映了设计者的数据全局逻辑要求，而外模式处于最外层，它反映了用户对数据的要求。

2. 数据库系统的两级映射

两级映射保证了数据库系统中数据的独立性。

（1）概念模式到内模式的映射。该映射给出了概念模式中数据的全局逻辑结构到数据的物理存储结构间的对应关系。

（2）外模式到概念模式的映射。概念模式是一个全局模式而外模式是用户的局部模式。一个概念模式中可以定义多个外模式，而每个外模式是概念模式的一个基本视图。

考点 5 数据模型的基本概念

数据模型用来抽象、表示和处理现实世界中的数据和信息。数据模型分为两个阶段：把现实世界中的客观对象抽象为概念模型；把概念模型转换为某一 DBMS 支持的数据模型。

数据模型所描述的内容有三个部分，它们是数据结构、数据操作与数据约束。

考点 6　E-R 模型

1. E-R 模型的基本概念

（1）实体：现实世界中的事物可以抽象为实体，实体是概念世界中的基本单位，它们是客观存在的且又能相互区别的事物。

（2）属性：现实世界中事物均有一些特性，这些特性可以用属性来表示。

（3）码：唯一标识实体的属性集称为码。

（4）域：属性的取值范围称为该属性的域。

（5）联系：在现实世界中事物间的关联称为联系。

两个实体集间的联系实际上是实体集间的函数关系，这种函数关系可以有下面几种：一对一的联系、一对多或多对一联系、多对多联系等。

2. E-R 模型的图示法

E-R 模型用 E-R 图来表示。

（1）实体表示法：在 E-R 图中用矩形表示实体集，在矩形内写上该实体集的名字。

（2）属性表示法：在 E-R 图中用椭圆形表示属性，在椭圆形内写上该属性的名称。

（3）联系表示法：在 E-R 图中用菱形表示联系，菱形内写上联系名。

考点 7　层次模型

满足下面两个条件的基本层次联系的集合为层次模型。

（1）有且只有一个节点没有双亲节点，这个节点称为根节点；

（2）除根节点以外的其他节点有且仅有一个双亲节点。

考点 8　关系模型

关系模型用表格结构表达实体集，用键表示实体间的联系。不仅可用关系描述实体本身，而且可用关系描述实体之间的联系。可直接表示多对多的联系，每个属性不可再分，建立在数学概念基础上，有较强的理论依据。

关系模型采用二维表来表示，二维表一般满足下面七个性质：

（1）二维表中元组个数是有限的——元组个数有限性；

（2）二维表中元组均不相同——元组的唯一性；

（3）二维表中元组的次序可以任意交换——元组的次序无关性；

（4）二维表中元组的分量是不可分割的基本数据项——元组分量的原子性；

（5）二维表中属性名各不相同——属性名唯一性；

（6）二维表中属性与次序无关，可任意交换——属性的次序无关性；

（7）二维表属性的分量具有与该属性相同的值域——分量值域的统一性。

在二维表中唯一标识元组的最小属性值称为该表的键或码。二维表中可能有若干个健，它们称为表的侯选码或侯选健。从二维表的所有侯选键选取一个作为用户使用的键，称为主键或主码。表 A 中的某属性集是某表 B 的键，则称该属性值为 A 的外键或外码。

关系操纵：数据查询、数据删除、数据插入、数据修改。

关系模型允许定义三类数据约束，它们是实体完整性约束、参照完整性约束以及用户定义的完整性约束。

考点9 关系代数

当对关系模型进行查询运算，涉及多种运算时，应当注意它们之间的先后顺序，因为有可能进行投影运算时，把符合条件的记录过滤，产生错误的结果。

1. 关系模型的基本操作

关系模型的基本操作：插入、删除、修改和查询。

其中查询包含如下运算：

（1）投影运算。从 R 中选择出若干属性列组成新的关系。

（2）选择运算。选择运算是一个一元运算，关系 R 通过选择运算（并由该运算给出所选择的逻辑条件）后仍为一个关系。设关系的逻辑条件为 F，则 R 满足 F 的选择运算可写成：σF（R）。

（3）笛卡尔积运算。设有 n 元关系 R 及 m 元关系 S，它们分别有 p、q 个元组，则关系 R 与 S 经笛卡尔积记为 $R\times S$，该关系是一个 $n+m$ 元关系，元组个数是 $p\times q$，由 R 与 S 的有序组组合而成。

2. 关系代数中的扩充运算

（1）交运算：关系 R 与 S 经交运算后所得到的关系是由那些既在 R 内又在 S 内的有序组所组成的，记为 $R\cap S$。

（2）除运算：如果将笛卡尔积运算看作乘运算的话，除运算就是它的逆运算。当关系 $T=R\times S$ 时，则可将除运算写成：$T\div R=S$ 或 $T/R=S$。

S 称为 T 除以 R 的商。除法运算不是基本运算，它可以由基本运算推导而出。

（3）连接与自然连接运算。连接运算又可称为 θ 运算，这是一种二元运算，通过它可以将两个关系合并成一个大关系。设有关系 R、S 以及比较式 $i\theta j$，其中 i 为 R 中的域，j 为 S 中的域，θ 含义同前。则可以将 R、S 在域 i，j 上的 θ 连接记为：

$$R\ |\times|\ S$$
$$i\theta j$$

在 θ 连接中如果 θ 为“=”，就称此连接为等值连接，否则称为不等值连接；如果 θ 为“<”时，称为小于连接；如果 θ 为“>”时，称为大于连接。

自然连接（natural join）是一种特殊的等值连接，它满足这样的条件：两关系间有公共域；通过公共域的等值进行连接。

设有关系 R、S，R 有域 A_1，A_2，…，A_n，S 有域 B_1，B_2，…，B_m，并且，A_{i1}，A_{i2}，…，A_{ij}，与 B_1，B_2，…，B_j 分别为相同域，此时它们自然连接可记为：

$$R|\times|S$$

自然连接的含义可用下式表示：

$$R|\times|S=\pi_{A1,A2,\cdots\cdots An,Bj+1,\cdots\cdots Bm}(\sigma_{Ai1=B1\wedge Ai2=B2\wedge\cdots\wedge Aij=,Bj}(R\times S))$$

考点 10　数据库设计概述

考试链接：

考点 10 在笔试考试中出现的概率为 30%，主要是以选择题的形式出现，分值为 2 分，此考点为识记内容，读者应识记数据库设计的前四个阶段以及它们相应的任务。

数据库设计中有两种方法，即面向数据的方法和面向过程的方法。

面向数据的方法是以信息需求为主，兼顾处理需求；面向过程的方法是以处理需求为主，兼顾信息需求。由于数据在系统中稳定性高，数据已成为系统的核心，因此面向数据的设计方法已成为主流。

数据库设计目前一般采用生命周期法，即将整个数据库应用系统的开发分解成目标独立的若干阶段。它们是需求分析阶段、概念设计阶段、逻辑设计阶段、物理设计阶段、编码阶段、测试阶段、运行阶段和进一步修改阶段。在数据库设计中采用前四个阶段。

全国计算机二级 C 语言程序设计考点

第一章　数据结构与算法

C 语言程序须注意：

（1）程序的构成，main（）函数和其他函数。

（2）头文件，数据说明，函数的开始和结束标志以及程序中的注释。

（3）源程序的书写格式。

（4）C 语言的风格。

考点 1　C 语言程序

考点点拨：重点记忆程序的构成和书写。

程序是可以连续执行的指令的集合。目前常用的程序语言主要是“高级语言”，如 Visual Basic、C++、Java 和 C。其中 C 语言具有高级语言和低级语言的双重优点。

C 语言程序由函数构成。一个 C 语言源程序有且仅有一个 main（）函数和零个或多个其他函数。无论 main（）函数的位置如何，它是程序执行的入口和出口。

C 语言程序书写自由，一行内可以写一条或多条语句，一条语句也可以写在多行。C 语言的注释有两种形式，分别为行尾注释“//……”和块式注释“/* …… */”。注释只是给用户看，对编译和运行不起作用。

考点 2　C 语言程序编译连接过程

考点点拨：C 程序的编译连接和后缀。

C 源程序文件（后缀为 .c）必须翻译为二进制目标文件（后缀为 .obj），此过程称为“编译”，负责此工作的程序称为“编译器”或“编译程序”；然后由连接程序把该二进制文件与 C 语言的各种库函数连接起来，生成可执行文件（后缀为 .exe），此过程称为“连接”；最后执行该可执行文件，实现程序功能。

程序设计指设计、编程、调试程序的方法和过程。程序设计通常分为问题建模、算法设计、编写代码和编译调试四个阶段。

第二章 数据类型、运算符和表达式

数据类型及其运算包括：

（1）C 语言的数据类型（基本类型、构造类型、指针类型、无值类型）及其定义方法。

（2）C 语言运算符的种类、运算优先级和结合性。

（3）不同类型数据间的转换与运算。

（4）C 语言表达式类型（赋值表达式、算术表达式、关系表达式、逻辑表达式、条件表达式、逗号表达式）和求值规则。

考点 1 标识符及命名规则

考点点拨：标识符的命名规则。

标识符的命名规则如下：

（1）由字母、数字、下划线三类字符组成；

（2）必须以字母或下划线打头；

（3）区分大小写；

（4）不能和关键字相同；

（5）尽量见名知义。

考点 2 常量

常量就是在程序运行中，值不能被改变的量。

常量的类型：整型常量、实型常量、字符常量、字符串常量、符号常量。

1. 整型常量

C 语言中整型常量的表示方法有十进制（以非零数字打头）、八进制（以数字 0 打头，后跟八进制数符）、十六进制（以 0X 或 0x 打头，后跟十六进制数符）。

C 语言中整型数据有三大类：短整型（short，2 B）、基本整型（int，4 B）、长整型（long，4 B），此为有符号类型，可表示正、负数；在三个关键字前加上 unsigned 则为无符号类型，所占字节数不变，但只能表示正数。

整型数据在内存中以二进制补码存放，若超出其表示范围，会产生溢出。

2. 实型常量

实型常量的表示方法有两种：小数形式和指数形式。

实型数据主要使用两种类型：单精度（float，4 B）、双精度（double，8 B），注意不同类型的有效数字的位数区别。

实型数据在内存中以指数形式存放，实型常量默认为 double 型，可以在常量后加 f 或 F，将其按 float 型处理。

3. 字符常量及转义字符常量

普通字符常量是用一对单引号括起来的一个字符表示的常量。字符常量的值是该字符的 ASCII 码值，如' A' 的值是 65，' 0' 字符的值是 48 等，在内存中占 1 B 空间。

转义字符常量指在单引号内由反斜线打头，后跟特定字符表示的常量。需要记的转义字符有三类：

（1）控制字符：' \ n' 、' \ r' 、' \ b' 、' \ t' 。

（2）特殊字符：' \ \ ' 、' \ '' 、' \ ” ' 。

（3）转义进制：' \ bbb' （1~3 位八进制数符）、' \ xhh' （1~2 位十六进制数符）。

字符串常量是用一对双引号括起来的零个或多个字符序列，其中包含的字符可以是普通字符或转义字符，系统自动在字符串末尾增加一个不可显示字符' \ 0' （空字符）作为字符串结束标记，字符串所占的内存空间为所含字符数+1。

考点 3　变量

考点点拨：变量的定义及赋值。

C 语言规定变量要先定义后使用，目的是为变量分配空间，同时为了编译时进行与操作相关的语法检查。

1. 整型变量

整型变量数据在内存中以二进制补码形式存放，其定义方式例如：

```
int a,b;
```

2. 实型变量

float 型数据在内存中占 4 个字节（32 位），double 型在内存中占 8 个字节（64 位）。

3. 字符型变量

每个字符型变量只能存放一个字符。其定义方式如：

```
char cr1,cr2;                    //指定 cr1,cr2 为字符型变量
char x=' a' ;                    //指定 x 为字符型变量,存储了字符‘a’
```

字符型数据与整型数据之间可以通用，一个字符能用字符的形式输出，也能用整数的形式输出。字符数据进行算术运算，相当于对它们的 ASCII 码进行运算。

考点 4　运算符的种类、优先级和结合性

考点点拨：运算符的分类、常用运算符的优先级和结合性。

运算符按功能分有 13 类，按所需操作数目分可分为单目运算符（右结合，优先级 2）、双目运算符（除赋值为右结合外，其余为左结合）、三目运算符（条件运算符，右结合）。

考点 5 不同类型数据间的混合运算

考点点拨：系统自动转换成同一类型的方向和结果的类型。

不同数据类型混合运算时系统自动转换的规则如图 4-3 所示，其中水平箭头为必然转换，垂直箭头为从低向高方向转换。运算结果类型为转换后的类型。

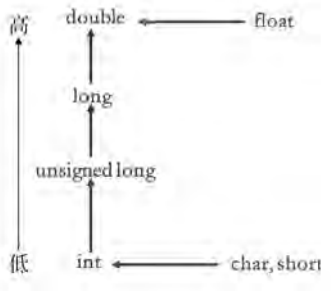


图 4-3 不同类型的混合运算转换规则

考点 6 算术运算符与表达式

考点点拨：重点记算术运算符各自的运算特点。

基本算术运算符+、-、*、/、%均为双目运算符。其中%运算符要求两侧的操作数必须为整型；/运算符两侧若均为整数，则结果为除以后的整数部分，否则为实型。

考点 7 赋值运算符与表达式

考点点拨：重点记赋值表达式及复合赋值表达式的特点。

赋值表达式左侧必须是简单变量，不能是常量或表达式（包括赋值表达式），右边可以是常量、变量或表达式（包括赋值表达式）。赋值运算符及复合赋值运算符右结合，优先级倒数第二。当赋值表达式右边表达式值的类型与左边变量的类型不一致时，系统会自动将该值转换成左边变量的类型后再赋值。

考点 8 自增、自减运算及表达式

考点点拨：理解前置运算和后置运算的不同意义。

自增（++）、自减（--）均为单目运算符，右结合，操作数可以是整型变量或实型变量，但不能是常量或表达式。前置运算（如++i、--i）是先增减后运算，后置运算（如 i++、i--）是先运算后增减。

考点 9 逗号运算及表达式

考点点拨：记逗号运算符的优先级、结合性及求值规则。

逗号表达式是用逗号运算符（,）将两个或两个以上的表达式顺序连接起来的表达式，左结合，逗号运算符优先级最低。从左到右顺序求解各表达式，也称为顺序求值表达式，

结果为最右边表达式的值。注意区分逗号分隔符和逗号运算符。

考点 10　类型转换

考点点拨：理解强制类型转换、自动类型转换。

（1）强制类型转换。格式为（类型）（表达式），将表达式值的类型强制转换成指定的类型，对表达式中原有变量的类型无影响。

（2）自动类型转换。

1）赋值运算中的自动转换。

2）混合运算中的自动转换。

考点 11　关系运算符及表达式

考点点拨：掌握关系运算符及表达式的求值。

关系运算符有：>、>=、<、<=、==、!=。[区分关系等（==）和赋值等（=）]

关系表达式用于判断两个对象之间的关系，其操作数可以是 C 语言中任何合法的表达式。关系表达式的值为逻辑值，关系成立为真，用整数 1 表示；关系不成立为假，用整数 0 表示。该值可以继续参加其他表达式运算。

注意：关系表达式不能直接判断三个以上对象的关系。

考点 12　逻辑运算符及表达式

考点点拨：掌握逻辑运算符及表达式的求值。

逻辑运算符有：!、&&、||（按优先级从高到低排列）。

逻辑运算符常结合关系表达式判断多个对象之间的复杂关系。逻辑表达式的值为逻辑值。参与逻辑运算的操作数可以是任意合法的常量、变量或表达式，均以“非零为真，零为假”判断其逻辑值。注意逻辑表达式求值时的“短路特性”。

考点 13　条件运算符及表达式

考点点拨：理解条件表达式的求值方法。

条件运算符（?　:）是 C 语言中唯一的一个三元运算符，由问号“?”和冒号“:”组成，连接三个运算对象，用来在两个表达式中选择一个表达式。表达形式如下：

表达式 1? 表达式 2：表达式 3；

表达式 1 是关系或布尔型，返回值为 bool 型；如果表达式 1 的值为 true，则整体表达式的值为表达式 2 的值；如果表达式 1 的值为 false，则整体表达式的值为表达式 3 的值。

考点 14　位运算符及表达式

考点点拨：理解位运算符及其求值规则和优先级顺序。

位运算符有：~、[<<、>>（同优先级）]、&、^、|（按优先级从高到低排列）。

位运算的操作数必须是整型或字符型，计算时先将操作数转换成二进制，然后再低位

对齐按运算规则进行计算。

第三章 控制结构

基本语句：

（1）表达式语句、空语句、复合语句。

（2）输入/输出函数的调用，正确输入数据并正确设计输出格式。

考点 1 C 语句的分类

考点点拨：理解 C 语句的概念及分类。

C 语句是函数体里的基本构成单位。语句的作用是向计算机系统发出操作指令，要求执行相应的操作。一个 C 语句经过编译后产生若干条机器指令。C 语句可以分成五大类：控制语句、函数调用语句、表达式语句、空语句和复合语句。

考点 2 putchar（）函数和 getchar（）函数

考点点拨：掌握字符数据的输入/输出函数的用法。

putchar 函数用于向显示器输出一个字符，一般格式是：putchar（c），其中 c 可以是字符型或整型常量、变量或表达式，结果是向显示器输出其值对应的 ASCII 码字符。

getchar 函数用于从键盘输入缓冲区读取一个字符，一般格式是 getchar（）。一般用法：ch=getchar（）;，将从键盘输入的一个字符值赋给变量 ch。从键盘输入的空格、回车等控制字符也作为有效字符读取。

考点 3 printf（）函数和 scanf（）函数

考点点拨：掌握格式化输出、输入函数的用法及注意事项。

printf（“格式控制字符串”,［输出表列］）;

scanf（“格式控制字符串”,［输入地址表列］）;

格式控制字符串中允许有两类字符：

（1）普通字符。在 printf（）函数中会原样输出，在 scanf（）函数中要求原样输入。

（2）格式字符。由“%+格式字符”，用于指定输出、输入项的格式。

考点 4 if 语句

考点点拨：掌握三种 if 语句的执行过程及 if 嵌套语句和 switch 语句。

常用三种形式是单分支、双分支和多分支。

1. 单分支 if 语句

单分支 if 语句的一般形式为：

if（表达式）〈语句〉;

2. 双分支 if 语句

双分支 if 语句的一般形式为：

if（表达式）〈语句 1〉；

else〈语句 2〉；

首先判断表达式的值，若表达式的值为“真”（非 0），则执行语句 1；否则，执行语句 2。

3. 多分支 if 语句

多分支 if 语句的一般形式为：

if（表达式 1）〈语句 1〉；

else if（表达式 2）〈语句 2〉；

…

else if（表达式 n）〈语句 n〉；

else〈语句 n+1〉；

从表达式 1 的值开始进行判断，当出现某个表达式的值为真时，则执行其对应分支的语句，然后跳出整个 if 语句，执行后续语句。若所有表达式的值都为“假”（为 0），则执行语句 n。

考点 5　switch 语句

考点点拨：掌握 switch 语句。

switch 语句又称开关语句，一般用于实现多分支选择。

注意：switch 后的表达式值的类型一般为整型或字符型。

case 子句后的常量必须为简单的整型或字符型常量。

当执行完一个分支的执行语句后没有遇到 break 时，将继续执行下一分支的执行语句。

switch 语句可以嵌套，break 在 switch 中的作用是跳出所在的 switch 语句。

考点 6　while 语句

考点点拨：理解 while 循环的执行过程及应用。

while 语句一般形式如下：

循环变量的初始值；

while（循环条件表达式）

循环体语句；

循环体语句可以是一条，也可以是多条，多条的时候应用“ {} ”将其括起来，使其成为复合语句。

考点 7　do…while 语句

该语句的特点是先执行循环体再判断循环条件，循环体至少执行一次。注意 while 之

后的分号不能少。

考点 8 for 语句

考点点拨：理解 for 循环的执行过程及应用。

for 语句的一般形式是：

for（表达式 1；表达式 2；表达式 3）

for 中的三个表达式可以是任意合法的 C 语言表达式，表达式 1 在进入循环时先执行一次，一般是给循环变量赋初值；表达式 2 取其逻辑值作为循环条件；表达式 3 在执行循环体后才执行，一般是使循环条件趋于假的运算。三个表达式均可以省略，但分号不能省。

考点 9 循环嵌套

考点点拨：理解循环嵌套的几种形式及执行过程。

循环体内又完整地包含了另一个循环，称循环嵌套。前三种循环语句可以相互嵌套，也可多层嵌套。嵌套循环在执行时外循环执行一次内循环要执行一遍。书写上一般采用缩进形式，使程序层次分明，可读性强。

考点 10 break 语句和 continue 语句

考点点拨：掌握 break 和 continue 语句在循环体中的应用。

break 语句的作用是退出所在的循环体，在循环体中使用可增加循环的出口，使循环更灵活。break 只能用于 switch 或循环语句中。

continue 语句的作用是结束本层本次的循环，转去执行下一次的循环处理。continue 语句只能用于循环体中。

第四章 函 数

考点 1 函数

考点点拨：理解函数在 C 语言中的功能，掌握如何调用库函数。

一个 C 程序有且仅有一个 main 函数，程序由 main 函数和若干个其他函数构成。主函数可以调用其他函数，其他函数可以相互调用。这些函数可以是库函数，也可以是用户自定义函数。

函数之间可以相互调用，但各函数必须是相互独立的，一个函数并不属于其他函数。其他函数不能调用 main 函数。C 语言系统提供了丰富的库函数，编程时可直接调用。

考点 2 函数的定义和声明

考点点拨：掌握函数定义和声明的一般形式。

一个函数必须定义后才能使用。所谓定义函数，就是编写完成函数功能的程序块。C语言函数由函数头与函数体两部分组成，其一般形式如下：

```
[<返回类型>]<函数名>（[形式参数列表]）    //函数头
{
    <函数体>                              //函数体
}
```

考点3　函数的形参和实参

考点点拨：掌握形参和实参的概念及作用。

定义函数时，函数名后的参数称为形参；调用函数时，函数名后的参数称为实参。当在一个函数中调用另一个函数时，前者称主调函数，后者称被调函数。主调函数通过实参向被调函数的形参传递数据。说明：

（1）实参可以是常量、变量、表达式或函数，形参只能是变量。

（2）实参和形参的类型一致或赋值兼容，个数必须相同，按顺序一一对应。

（3）当函数被调用时，形参才分配内存空间，调用结束时，形参所占内存空间被释放。

（4）实参对形参的数据传递是单向值传递，而且在内存中，形参与实参占用的是不同的内存单元，因此形参的改变并不影响实参。

考点4　函数的值

考点点拨：理解在函数调用时 return 语句的作用。

根据函数的不同功能，将C语言的函数分为两类，一类函数用于计算一个值，称为具有返回值的函数；另一类函数仅仅是为了实现一个过程，而不是为了得到一个值，称为无返回值的函数。

被调函数可以用 return 语句将函数值传递给主调函数。return 语句形式如下：

return 表达式；　或　return（表达式）；

说明：

（1）当函数执行到 return 语句时，返回到它的主调函数的调用位置，并带回返回值。

（2）return 后的表达式可以是常量、变量或表达式。

（3）表达的类型若和函数定义中的返回值类型不相同，则系统自动转换为定义的类型；若无法转换，则赋值不兼容。

（4）若函数定义为 void 类型，则不能用 return 带回返回值。函数最后一个“}”起返回作用。

（5）函数中可以有多条 return 语句，但只执行其中一条，或都不执行。

考点5　函数的调用

考点点拨：掌握函数调用的基本形式及其语法要求。

使用已经定义的函数的过程，称为函数的调用。函数的调用方式可以分为一般调用、嵌套调用和递归调用三类。函数调用的形式如下：

函数名（实参表）

被调用的函数中又调用另外一个函数，称嵌套调用。

函数直接或间接地调用自身，称递归调用。

考点6 变量的作用域

考点点拨：掌握局部变量和全局变量的作用范围及特点。

C语言将变量分为局部变量和全局变量。

在一个函数内部声明的或在一个块中定义的变量是局部变量，其作用域只在本函数范围内。即局部变量只能在定义它的函数体内部使用，而不能在其他函数中使用。

在函数外部定义的变量称为全局变量，可被作用域内的所有函数直接引用。

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。

当全局变量和局部变量同名时，在局部变量的作用范围内同名的全局变量不起作用。

考点7 宏定义

在C程序中，用一个标识符来表示一个字符串，称为“宏”，被定义为“宏”的标识符则称为“宏名”。宏定义可分为两种形式，带参数的宏定义和不带参数的宏定义。

考点8 文件包含

考点点拨：理解文件包含的作用、形式及特点。

“文件包含”处理是指一个源文件可以将另外一个源文件的全部内容包含进来。C语言提供了#include命令用来实现“文件包含”的操作。一般形式为：

```
#include "文件名"
```

或 #include <文件名>

第五章 数 组

考点1 一维数组

考点点拨：掌握一维数组的定义方式及初始化的定义形式。

一维数组的定义方式：

数据类型 数组名［常量表达式］；

常量表达式规定了数组元素的个数（或长度），整个数组所占字节数=类型长度×数组长度；常量表达式中可以包括整型常量和符号常量；在给全部元素赋初值时可以省略定义时的数组长度说明。

一维数组初始化的一般形式：

数据类型　数组名［常量表达式］＝｛值0，值1，值2，……｝；

初值的个数不能超过数组的长度；可以只给部分元素赋初值，未赋初值元素默认值为0。

考点2　二维数组

考点点拨：掌握二维数组的定义方式及初始化的定义形式。

二维数组的定义方式：

数据类型　数组名［常量表达式1］［常量表达式2］；

二维数组一般用于存储矩阵，常量表达式1为矩阵行数，常量表达式2为矩阵列数。二维数组元素在内存中是按行存放的，各元素在内存中所占的字节数＝行数×列数×类型长度。

考点3　字符数组

考点点拨：掌握字符数组的定义及初始化形式。

在C语言中只有字符变量，没有字符串变量，通常用字符数组来存储字符串。字符数组就是类型为char的数组，用于存储一串连续的字符。字符数组中每个数组元素存放的值都是单个字符（1个字节），这些单个字符连续存储就构成了字符数组，字符数组既可以是一维的，也可以是多维的。

考点4　字符串处理函数

考点点拨：掌握各常用的字符串处理函数的应用。

常用的字符串处理函数如下（须包含string.h头文件）：

strcpy（s1，s2）——字符串拷贝函数：将s2中的内容复制到字符数组s1中；

strncpy（s1，s2，n）——拷贝子串函数：将s2中的前n个字符拷贝到s1中；

strcat（s1，s2）——字符串连接函数：将s2连接在s1的末尾；

strcmp（s1，s2）——字符串比较函数：从左到右逐个字符比较两个字符串的大小；

strlen（str）——求字符函数串长度：求字符数组str中第一个'\0'字符之前的字符个数。

第六章　指　　针

考点1　变量地址和指针变量的概念及赋值

考点点拨：理解变量的地址和指针的概念，掌握指针变量的定义和赋值。

地址是内存空间某一字节的编号，变量的地址是指给变量在内存中分配的空间的起始地址编号。

指针变量是专门存放变量地址的一种特殊变量。

C 语言允许可以用变量名直接访问所分配的内存空间，也可以通过指针变量间接访问所指向变量的内存空间。

考点 2 通过指针变量引用存储单元

考点点拨：掌握如何通过指针变量间接访问所指地址空间。

指针变量在运算时常用两个运算符：

（1）&：取地址运算符，可取变量的地址、数组元素的地址，如 &i，&a［i］。

（2）*：间接访问运算符，*右边的运算对象可以是指针变量或变量的地址。

考点 3 指针变量作函数参数

考点点拨：掌握指针变量作函数参数传递和返回值的意义。

指针可以作为函数的形参。这样，函数调用时实际是将实参的地址传递给形参，即形参指针变量指向实参，这种传递方式叫地址值传递。地址值传递方式的特点是可以通过对形参指针变量的间接访问引用、改变实参变量的值。这样调用函数可以实现不用 return 而得到多个函数处理结果，该方式在程序中广泛使用。

考点 4 指向数组的指针

考点点拨：掌握指向一维数组、二维数组和字符数组的指针的应用。

考点 5 数组的指针作函数参数

考点点拨：掌握数组的指针作函数参数的特点和应用。

数组作函数参数有以下情况：

（1）数组元素作函数参数：同普通变量作函数参数，此时形参为普通变量，是单向普通值传递，最多由 return 带回一个返回值。

（2）一维数组名作函数参数：传递的是一维数组的首地址，此时形参为一级指针变量或同类型的一维数组。可以在被调函数中通过形参变量或数组名间接访问、改变所有的实参数组元素值。

（3）二维数组名作函数参数：传递的是二维数组的首行地址，此时形参为二级指针变量或同类型二维数组。

第七章 结构体和共用体

考点 1 结构体类型及变量的定义

考点点拨：理解结构体类型的声明及变量的定义。

声明结构体类型用关键字 struct，形式如下：

struct　　［结构体名］

｛成员列表　｝；

其中，“结构体名”是合法标识符，可以省略。“成员列表”定义该类型中各成员的类型和名字。结构体类型定义只描述结构中各成员的组织形式，各成员并不占用内存空间。可以嵌套定义结构体类型。

考点 2　结构体变量的赋值和引用

考点点拨：掌握结构体变量的赋值和引用。

可以在定义结构体变量时按成员顺序给其赋初值，如：

struct 结构体名　变量名 = ｛初值 1，初值 2，……｝；

除了同类型结构体变量可以整体赋值外，其余只能通过成员运算符（.）逐个引用其成员，也可以通过结构体指针引用其指向变量的成员。

考点 3　用结构体处理链表

考点点拨：理解链表的结构特点及链表的建立、输出、插入、删除。

链表是一种动态存储结构，其基本构成单位称节点，其特点是通过节点中的指针来体现节点间的先后关系。节点中一般包含两部分数据：数据域和指针域。

考点 4　共用体

考点点拨：了解共用体类型和变量的定义及引用方法。

共用体类型用关键字 union 定义，其形式同结构体类型。

共用体变量的定义和引用同结构体变量，区别在于：

（1）共用体变量分配的字节数为各成员所占字节数的最大值；

（2）共用体变量不能在定义的时候赋初值；

（3）共用体变量中各成员不能同时存在，某一时刻只能有一个成员起作用且是最后一次赋值的成员。

第八章　文　　件

考点 1　文件的基本概念及分类

考点点拨：了解 C 程序处理文件的分类。

在 C 程序中处理的文件通常为提供原始数据或存放结果数据，因其存放在外存上便于长期保存和与程序数据共享。C 程序处理的文件分为两种：文本文件和二进制文件。

考点 2　文件类型和打开、关闭方法

考点点拨：掌握文件类型及文件打开、关闭函数的使用方法。

文件类型是在 stdio.h 文件中定义的 FILE 结构体类型，通常使用 FILE 定义文件类型指针，用该指针保存文件打开的返回值，并通过该指针完成对文件的读写和其他操作。

文件的操作顺序是：打开 →读写 →关闭。

打开文件用 fopen 函数，一般用法是：fp=fopen（“文件名”“打开方式”）；打开成功后建立文件指针 fp 和所打开文件的指向关系，打开失败返回 NULL。

文件的打开方式有基本的三种：r（只读）、w（只写）、a（追加），扩展后共 12 种，根据操作需要选择合适的打开方式。

文件关闭用 fclose 函数，一般用法是：fclose（fp）;。

考点 3 文件字符的输入/输出

考点点拨：了解文件字符输入/输出函数的使用。

从文件读取一个字符函数的用法一般是：

ch = fgetc（fp）;

函数调用成功返回从文件读取到的一个字符，读取失败返回 EOF（-1）。

输出一个字符到文件函数的一般用法是：

fputc（ch，fp）;

该函数通常针对以文本方式打开的文件。

从文件读取一个字符串函数的用法一般是：

fgets（str，N，fp）;

输出一个字符串到文件函数的一般用法是：

fputs（str，fp）;

该函数通常针对以文本方式打开的文件。

考点 4 文件的读写

考点点拨：了解文件的格式化输入/输出函数的使用。

从文件读取指定格式数据函数的用法一般是：

（1）fwrite（）函数。一般格式：

fwrite（buffer，size，count，fp）;

（2）fread（）函数。一般格式：

fread（buffer，size，count，fp）;

（3）fprintf（）函数。一般格式：

fprintf（文件类型指针，格式控制，输出表列）;

（4）fscanf（函数）。一般格式：

fscanf（文件类型指针，格式控制，地址表列）;

该函数操作的可以是文本文件或二进制文件。

考点 5　文件的定位

（1）用 rewind（）函数使文件位置标记指向文件开头。一般形式：

rewind（文件指针）；

rewind（）函数的作用是使文件位置标记重新返回文件的开头，此函数没有返回值。

（2）用 fseek 函数改变文件位置标记。一般形式：

fseek（文件类型指针，位移量，起始点）；

全国计算机等级考试二级 C 语言样题

一、选择题（（1）～（10）、（21）～（50）每题 1 分，（11）～（20）每题 2 分，共 60 分）

（1）下列选项中不符合良好程序设计风格的是_____。

A. 源程序要文档化　　B. 数据说明的次序要规范化

C. 避免滥用 goto 语句　　D. 模块设计要保证高耦合、高内聚

（2）从工程管理角度，软件设计一般分为两步完成，它们是_____。

A. 概要设计与详细设计　　B. 数据设计与接口设计

C. 软件结构设计与数据设计　　D. 过程设计与数据设计

（3）下列选项中不属于软件生命周期开发阶段任务的是_____。

A. 软件测试　　B. 概要设计　　C. 软件维护　　D. 详细设计

（4）在数据库系统中，用户所见的数据模式为_____。

A. 概念模式　　B. 外模式　　C. 内模式　　D. 物理模式

（5）数据库设计的四个阶段是：需求分析、概念设计、逻辑设计和_____。

A. 编码设计　　B. 测试阶段　　C. 运行阶段　　D. 物理设计

（6）设有如下三个关系表　下列操作中正确的是_____。

R

A
m
n

S

B	C
1	3

T

A	B	C
m	1	3
n	1	3

A. T=R∩S　　B. T=R∪S　　C. T=R×S　　D. T=R/S

（7）下列叙述中正确的是_____。

A. 一个算法的空间复杂度大，则其时间复杂度也必定大

B. 一个算法的空间复杂度大，则其时间复杂度必定小

C. 一个算法的时间复杂度大，则其空间复杂度必定小

D. 上述三种说法都不对

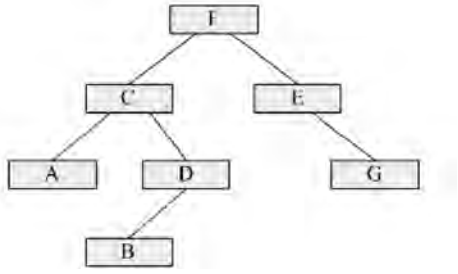
（8）在长度为 64 的有序线性表中进行顺序查找，最坏情况下需要比较的次数为_____。

A. 63　　B. 64　　C. 6　　D. 7

（9）数据库技术的根本目标是要解决数据的_____。

A. 存储问题　　B. 共享问题　　C. 安全问题　　D. 保护问题

（10）对下列二叉树进行中序遍历的结果是_____。



A. ACBDFEG　　B. ACBDFGE　　C. ABDCGEF　　D. FCADBEG

（11）下列叙述中错误的是_____。

A. 一个 C 语言程序只能实现一种算法

B. C 程序可以由多个程序文件组成

C. C 程序可以由一个或多个函数组成

D. 一个 C 函数可以单独作为一个 C 程序文件存在

（12）下列叙述中正确的是_____。

A. 每个 C 程序文件中都必须有一个 main（）函数

B. 在 C 程序中 main（）函数的位置是固定的

C. C 程序可以由一个或多个函数组成

D. 在 C 程序的函数中不能定义另一个函数

（13）下列定义变量的语句中错误的是_____。

A. int _int;　　B. double int_;　　C. char For;　　D. float USS

（14）若变量 x、y 已正确定义并赋值，以下符合 C 语言语法的表达式是_____。

A. ++x，y=x--　　B. x+1=y　　C. x=x+10=x+y　　D. double（x）/10

（15）以下关于逻辑运算符两侧运算对象的叙述中正确的是_____。

A. 只能是整数 0 或 1　　B. 只能是整数 0 或非 0 的整数

C. 可以是结构体类型的数据　　D. 可是任意合法的表达式

（16）若有定义 int x，y；并已正确给变量赋值，则以下选项中与表达式（x-y)?（x++)：(y++）中的条件表达式（x-y）等价的是_____。

A. （x-y>0）　　B. （x-y<0）

C. （x-y<|| x-y>0）　　D. （x-y==0）

（17）有以下程序：

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

程序运行后的输出结果是_____。

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

（18）设有定义：int a；float b；执行 scanf（"%2d%f"，&a，&b)；语句时，若从键盘输入 876 543. 0<回车>，a 和 b 的值分别是_____。

A. 876 和 543. 000000　　B. 87 和 6. 000000

C. 87 和 543. 000000　　D. 76 和 543. 000000

（19）有以下程序：

```
main（）
{int a=0，b=0;
a=10;/*给 a 赋值 */
b=20;/*给 b 赋值 */
printf（"a+b=%d\n"，a+b);/* 输出计算结果 */}
```

程序运行后输出结果是_____。

A. a+b=0　　B. a+b=30　　C. 30　　D. 出错

（20）在嵌套使用 if 语句时，C 语言规定 else 总是_____。

A. 和之前与其具有相同缩进位置的 if 配对

B. 和之前与其最近的 if 配对

C. 和之前与其最近的且不带 else 的 if 配对

D. 和之前的第一个 if 配对

（21）下列叙述中正确的是_____。

A. break 语句只能用于 switch 语句

B. 在 switch 语句中必须使用 default

C. break 语句必须与 switch 语句中的 case 配对使用

D. 在 switch 语句中，不一定使用 break 语句

（22）有以下程序：

```
main（）
{int k=5;
while（-k）
printf（"%d"，k-=3);
printf（"\n");}
```

执行后的输出结果是_____。

A. 1　　B. 2　　C. 4　　D. 死循环

(23) 有以下程序：

```
main ()
{int i;
for (i=1; i<=40; i++)
{if (i++%5=0)
if (++i%8=0) printf ("%d", i); }
printf ("\n"); }
```

执行后的输出结果是_____。

A. 5　　B. 24　　C. 32　　D. 40

(24) 以下选项中，值为 1 的表达式是_____。

A. 1-"0"　　B. 1-"\0"　　C. "1"-0　　D. "\0"-"0"

(25) 有以下程序：

```
fun (int x, int y) {return (x+y);}
main ()
{int a=1, b=2, c=3, sum;
sum=fun ( (a++, b++b, a+b), c++);
printf ("%d\n", sum); }
```

执行后的输出结果是_____。

A. 6　　B. 7　　C. 8　　D. 9

(26) 有以下程序：

```
main ()
{ char s [ ] = "abcde";
s+=2;
printf ("%d\n", s [0] ); }
```

执行后的输出结果是_____。

A. 输出字符 a 的 ASCII 码　　B. 输出字符 c 的 ASCII 码

C. 输出字符 c　　D. 程序出错

(27) 有以下程序：

```
fun (int x, int y)
{static int m=0, i=2;
i+=m+1; m=i+x+y;
return m;    }
main ()
{int j=1, m=I, k;
```

```
k=fun（j，m）;
printf（"%d"，k）;
k=fun（j，m）;
printf（"%d\n"，k）;}
```

执行后的输出结果是_____。

A. 5，5　　B. 5，11　　C. 11，11　　D. 11，5

（28）有以下程序：

```
fun（int x）
{int p;
if（x==0 ‖ x= =1）
return（3）;
p=x-fun（x=2）;
retum p;
} main（）
{ prinf（"%d\n"，fun（7））;}
```

执行后的输出结果是_____。

A. 7　　B. 3　　C. 3　　D. 0

（29）在16位编译系统上，有定义 int a［ ］ = {10，20，30}，*p=&a;，在执行 p++; 后，下列说法错误的是_____。

A. p 向高地址移了一个字节　　B. p 向高地址移了一个存储单元

C. p 向高地址移了两个字节　　D. p 与 a+1 等价

（30）有以下程序：

```
main（）
{int a=1，b=3，c=5;
int *p1=&a，*p2=&b，*p=&c;
*p=*p1*（*p2）;
printf（"%d\n"，c）;}
```

执行后的输出结果是_____。

A. 1　　B. 2　　C. 3　　D. 4

（31）若有定义：int w［3］［5］;，则以下不能正确表示该数组元素的表达式是_____。

A. *（*w+3）　　B. *（w+1）［4］

C. *（*（w+1））　　D. *（&w［0］［0］+1）

（32）若有以下函数首部 int fun（double x［10］，int *n），则下面针对此函数声明语句中正确的是_____。

A. int fun（double x，int *n）;　　B. int fun（double，int）;

C. int fun（double*x，int n）;　　D. int fun（double *，int*）;

（33）若有定义语句：int k［2］［3］，*pk［3］;，则以下语句中正确的是_____。

A. pk=k;　　B. pk［0］=&k［1］［2］;

C. pk=k［0］;　　D. pk［1］=k;

（34）有以下程序：

```
void change（int k［ ］）
{k［0］=k［5］;}
main（）
{int x［10］ = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}, n=0;
while（n<=4）
{change（&x［n］）; n++;}
for（n=0; n<5; n++）
printf（"%d", x［n］）; printf（"\n"）;　}
```

程序运行后输出的结果是_____。

A. 678910　　B. 13579　　C. 12345　　D. 62345

（35）若要求定义具有 10 个 int 型元素的一维数组 a，则以下定义语句中错误的是_____。

A. #define N 10 Int a［N］　　B. #define n 5 Int a［2*n］

C. int a［5+5］　　D. int n=10, a［n］

（36）有以下程序：

```
main（）
{int x［3］［2］ = {0}, I;
for（I=0; I<3; I++）
scanf（"%d", x［i］）;
printf（"%3d%3d%3d\n", x［0］［0］, x［0］［1］, x［1］［0］）;}
```

若运行时输入：246<回车>，则输出结果为_____。

A. 2 0 0　　B. 2 0 4　　C. 2 4 0　　D. 2 4 6

（37）有以下程序：

```
main（）
{char s［ ］ = { "aeiou"}, *ps;
ps=s;
printf（"%c\n", *ps+4）;}
```

程序运行后的输出结果是_____。

A. a　　B. e　　C. u　　D. 元素 s［4］的地址

（38）以下语句中存在语法错误的是_____。

A. char ss［6］［20］; ss［1］ = "right?";

B. char ss［］［20］ = { "right?"};

C. char *ss［6］; ss［1］ = "right?";

D. char *ss[] = {"right?"};

（39）若有定义：char *x="abcdefghi";，以下选项中正确运用了 strepy 函数的是_____。

A. char y[10]; strcpy（y, x[4]）;

B. char y[10]; strcpy（++y, &x[1]）;

C. char y[10], *s; strcpy（s=y+5, x）;

D. char y[10], *s; strcpy（s=y+1, x+1）;

（40）有以下程序：

```
int add（int a, int b）
{return+b};}
main（）
{int k,（*f）（）, a=5, b=10;
f=add;
… }
```

则以下函数调用语句错误的是_____。

A. k=（*f）（a, b）;　　B. k=add（a, b）;

C. k=*f（a, b）;　　D. k=f（a, b）;

（41）有以下程序：

```
#include <string. h>
main（int argc, char *argv[]）
{int i=1, n=0;
while（i<arge）
{n=n+strlen（argv[i]）; i++;}
printf（"%d\n", n）; }
```

该程序生成的可执行文件名为：proc. exe。若运行时输入命令行：proc 123 45 67，则程序的输出结果是_____。

A. 3　　B. 5　　C. 7　　D. 11

（42）有以下程序：

```
void fun2（char a, char b）
{printi（"%b%c", a, b）;}
char a='A', b='B';
void fun1（）{ a='C'l b='D'; }
main（）
{ fun1（）
printf（"%c%c", a, b）; fun2（'E', 'F'）;　}
```

程序的运行结果是_____。

A. CDEF　　B. ABEF　　C. ABCD　　D. CDAB

（43）有以下程序：

```
#include <stdio. h>
#define N 5
#define M N+1
#define f（x）（x * M）
main（）
{int i1, i2;
i1=f（2）;
i2=f（1+1）;
printf（"%d %d \ n", i1, i2）;  }
```

程序的运行结果是_____。

A. 12　12　　B. 11　7　　C. 11　11　　D. 12　7

（44）设有以下语句 typedef struct TT {char c; int a［4］;} CIN；则下面叙述中正确的是_____。

A. 可以用 TT 定义结构体变量　　B. TT 是 struct 类型的变量

C. 可以用 CIN 定义结构体变量　　D. CIN 是 struct TT 类型的变量

（45）有以下结构体说明、变量定义和赋值语句：

```
struct STD {char name［10］;
int age; char sex;
} s［5］, *ps; ps=&s［0］;}
```

则以下 scanf（）函数调用语句中错误引用结构体变量成员的是_____。

A. scanf（"%s", s［0］. name）;　　B. scanf（"%d", &s［0］. age）;

C. scanf（"%c", &（ps>sex））;　　D. scanf（"%d", ps>age）;

（46）若有以下定义和语句：

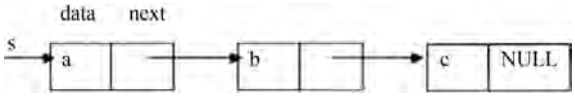
```
union data { int i; char c; float f; x; int y;}
```

则以下语句正确的是_____。

A. x=10. 5;　　B. x. c=101;

C. y=x;　　D. printf（"%d \ n", x）;

（47）程序中已构成如下图所示的不带头节点的单向链表结构，指针变量 s、p、q 均已正确定义，并用于指向链表节点，指针变量 s 总是作为头指针指向链表的第一个节点。



NULL 若有以下程序段：

```
q=s; s=s>next; p=s;
while（p->next）p=p->next;
p->next=q; q->next=NULL;
```

该程序段实现的功能是_____。

A. 首节点成为尾节点　　B. 尾节点成为首节点

C. 删除首节点　　D. 删除尾节点

（48）若变量已正确定义，则以下语句的输出结果是_____。

```
s=32; s^=32;
printf ("%d", s);
```

A. -1　　B. 0　　C. 1　　D. 32

（49）以下叙述中正确的是_____。

A. C 语言中的文件是流式文件，因此只能顺序存取数据

B. 打开一个已存在的文件并进行了写操作后，原有文件中的全部数据必定被覆盖

C. 在一个程序中当对文件进行了写操作后，必须先关闭该文件然后再打开，才能读到第 1 个数据

D. 当对文件的读（写）操作完成之后，必须将它关闭，否则可能导致数据丢失

（50）有以下程序：

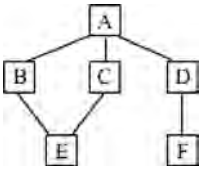
```
#include <stdio. h>
main ( )
{FILE *fp; int i; char ch [ ] ="abcd", t;
fp=fopen ("abc. dat","wb+");
for (i=0; i<4; i++) fwriter&ch [ ], 1, 1fp;
fseek (fp, -2L, SEEK_END);
fread (&t, 1, 1, fp);
fclose (fp);
printf ("%c \ n", t);  }
```

程序执行后的输出结果是_____。

A. d　　B. c　　C. b　　D. a

二、填空题（每空 2 分，共 40 分）

（1）下列软件系统结构图的宽度为_____。



（2）_____的任务是诊断和改正程序中的错误。

（3）一个关系表的行称为_____。

（4）按“先进后出”原则组织数据的数据结构是_____。

（5）数据结构分为线性结构和非线性结构，带链的队列属于_____。

（6）设有定义：float x=123. 4567;，则执行以下语句后的输出结果是_____。

```
printf ("%f\n", (int) (x*100+0.5) /100.0);
```

(7) 以下程序运行后的输出结果是_____。

```
main ()
{  int m=011, n=11;
printf ("%d  %d\n", ++m, n++);}
```

(8) 以下程序运行后的输出结果是_____。

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

(9) 有以下程序，若运行时从键盘输入：18，11<回车>，则程序输出结果是_____。

```
main ()
{  int a, b;
printf ("Enter a, b:");
scanf ("%d,%d", &a, &b);
while (a! =b)
   {  while (a>b) a-=b;
while (b>a) b-=a; }
printf ("%3d%3d\n", a, b);}
```

(10) 以下程序的功能是将输入的正整数按逆序输出。例如，若输入 135，则输出 531。请填空。

```
#include <stdio.h>
main ()
{int n, s;
printf ("Enter a number:");
scanf ("%d", &n);
printf ("Output:");
do
   { s=n%10;
printf ("%d", s); _____;}
while (n! =0);
printf ("\n");   }
```

(11) 以下程序中，函数 fun 的功能是计算 x^2-2x+6，主函数中将调用 fun 函数计算：

$$\begin{cases} y1=(x+8)\ 2-2\ (x+8)\ +6 \\ y2=\sin 2\ (x)\ -2\sin\ (x)\ +6 \end{cases}$$

请填空。

```
#include  "math. h"
double fun (double  x) { return (x * x-2 * x+6); }
main ()
{ double  x, y1, y2;
printf ("Enter x:");      scanf ("%lf", &x);
y1=fun (_____);
y2=fun (_____);
printf ("y1=%lf, y2=%lf\ n", y1, y2);
}
```

（12）下面程序的功能是：将 *N* 行 *N* 列二维数组中每一行的元素进行排列，第 0 行从小到大排序，第 1 行从大到小排序，第 2 行从小到大排序，第 3 行从大到小排序。请填空。

```
#define  N  4
void sort (int a [] [N] )
{int i, j, k, t;
for (i=0; i<N; i++)
for (j=0; j<N-1; j++)
for (k=_____; k<N; k++) /* 判断下标是否为偶数来确定按升序或降序来
                                排列 */
if (_____? a [i] [j] <a [j] [k]: a [i] [j] >a [i] [k] )
   { t=a [i] [j];
a [i] [j] =a [i] [k];
a [i] [k] =t; } }
void outarr (int a [N] [N] )
{.....}
main ()
{int aa [N] [N] = { {2, 3, 4, 1}, {8, 6, 5, 7}, {11, 12, 10, 9}, {15, 14,
16, 13} };
outarr (aa); /* 以矩阵的形式输出二维数组 */
  sort (aa);
outarr (aa);}
```

（13）下列程序中的函数 strcpy2 () 实现字符串两次复制，即将 t 所指字符串复制两次到 s 所指内存空间中，合并后形成一个新的字符串。例如，若 t 所指字符串为 efgh，调

用 strcpy2 后，s 所指字符串为 efghefgh。请填空。

```
#include <stdio.h>
#include <string.h>
void strcpy2 (char *s, char *t)
{ char *p=t;
while (*s++=*t++);
s=_____;
while (_____=*p++);}
main ()
{ char str1[100]="abcd", str2[]="efgh";
strcpy2 (str1, str2);
printf ("%s\n", str1);}
```

(14) 下面程序的运行结果是_____。

```
#include <stdio.h>
int f (int a[], int n)
{if (n>1)
return a[0]+f (a+1, n-1);
  else
return a[0];}
main ()
{int aa[10]={1, 2, 3, 4, 5, 6, 7, 8, 9, 10}, s;
s=f (aa+2, 4);
printf ("%d\n", s);}
```

(15) 下面程序由两个源文件 t4.h 和 t4.c 组成，程序编译运行的结果是_____。

t4.h 的源程序为：

```
  #define  N  10
#define  f2 (x) (x*N)
```

t4.c 的源程序为：

```
#include <stdio.h>
#define  M  8
#define  f (x) ( (x) *M)
#include "t4.h"
  main ()
{int i, j;
i=f (1+1);
```

```
j=f2（1+1）; printf（"%d%d\n", i, j）;    }
```

（16）下面程序的功能是建立一个有 3 个节点的单向循环链表，然后求各个节点数值域 data 中数据的和，请填空。

```
#include <stdio.h>
#include <stdlib.h>
struct NODE {
int data;
  struct NODE *next;    };
main（）
{struct NODE *p, *q, *r;
int sum=0;
p=（struct NODE*）malloc（sizeof（struct NODE））;
q=（struct NODE*）malloc（sizeof（struct NODE））;
r=（struct NODE*）malloc（sizeof（struct NODE））;
p->data=100;
q->data=200;
r->data=200;
  p->data=q;
q->data=r;
r->data=p;
m=p->data+p->next->data+r->next->data
_____;
  printf（"%d\n", sum）;    }
```

（17）有以下程序，其功能是以二进制“写”方式打开文件 d1.dat，写入 1～100 这 100 个整数后关闭文件。再以二进制“读”方式打开文件 d1.dat，将这 100 个整数读入另一个数组 b 中，并打印输出，请填空。

```
#include <stdio.h>
main（）
{ FILE *fp;
int i, a［100］, b［100］;
  fp=fopen（"d1.dat","wb"）;
for（i=0; i<100; i++）
a=i+1;
  fwrite（a, sizeof（int）, 100, fp）;
fclose（fp）;
  fp=fopen（"d1.dat", _____）;
```

```
    fread (b, sizeof (int), 100, fp);
fclose (fp);
for (i=0; i<100; i++)
printf ("%d\n", b);}
```

参考答案：

一、选择题

(1) ~ (10)：DACBD　CDBBA

(11) ~ (20)：AADDD　CCBBC

(21) ~ (30)：DACBC　DBCAC

(31) ~ (40)：ADBAD　BBADC

(41) ~ (50)：CABCD　BABDB

二、填空题

(1) 3　(2) 程序调试　(3) 元组　(4) 栈　(5) 线性结构

(6) 0.000000　(7) 10　11　(8) 1　(9) 7　4　(10) n=n/10

(11) (x+8), sin (x)　(12) 0, i%2==0　(13) s--, *s++　(14) 18

(15) 16　11　(16) ->data　(17)"rb"

参 考 文 献

[1] 谭浩强. C 语言设计学习辅导 [M]. 北京：清华大学出版社，2010.

[2] 张玉生，朱苗苗，张书月. C 语言程序设计实训教程 [M]. 上海：上海交通大学出版社，2018.

[3] 秦玉平，马靖善，王丽君. C 语言程序设计学习与实验指导 [M]. 3 版. 北京：清华大学出版社，2017.

[4] 孙辉. C 语言程序设计实验指导与习题集 [M]. 石家庄：中国铁道出版社，2016.

[5] 陈维. C 语言程序设计实训教程 [M]. 北京：人民邮电出版社，2018.