达州市职业高级中学校本教材

C语言程序设计

主编：杨从林

高等教育出版社

项目一 C 语言概述

C 语言是国际上广泛流行的计算机高级语言，既可用来写系统软件，也可用来写应用软件。

C 语言概念少，词汇少，包含了基本的编程元素，后来的很多语言（C++、Java 等）都参考了 C 语言，说 C 语言是现代编程语言的开山鼻祖毫不夸张，它改变了编程世界。正是由于 C 语言的简单，对初学者来说，学习成本小，时间短，能够快速掌握编程技术。

任务 1.1 认识 C 语言

任务目标

1.了解 C 语言的发展和 C 语言的特点

2. 了解 C 语言的程序结构

任务实施

1.1.1 C 语言的历史背景

C 语言是在 B 语言的基础上发展起来的，它的根源可以追溯到 ALGOL60。1960 年出现的 ALGOL60 是一种面向问题的高级语言，它离硬件比较远，不宜用来编写系统程序。1963 年英国的剑桥大学推出了 CPL(combined programming language)语言。CPL 语言在 ALGOL60 的基础上接近硬件一些，但规模比较大，难以实现。1967 年英国剑桥大学的 MatinRichards 对 CPL 语言做了简化，推出了 BCPL(basic combined programming language)语言。1970 年美国贝尔实验室的 KenThompson 以 BCPL 语言为基础，又做了进一步简化，设计出了很简单的而且很接近硬件的 B 语言(取 BCPL 的第一个字母)，并用 B 语言写了第一个 UNIX 操作系统，在 PDP7 上实现。1971 年在 PDP11/20 上实现了 B 语言，并写了 UNIX 操作系统。但 B 语言过于简单，功能有限。

1972 年至 1973 年间，贝尔实验室的 D.M.Ritchie 在 B 语言的基础上设计出了 C 语言(取 BCPL 的第二个字母)。C 语言既保持了 BCPL 和 B 语言的优点(精练，接近硬件)，又克服了它们的缺点(过于简单，数据无类型等)。最初的 C 语言只是为描述和实现 UNIX 操作系统提供一种工作语言而设计的。1973 年，K.Thompson 和 D.M.Ritchie 两人合作把 UNIX 的 90%以上用 C 改写，即 UNIX 第 5 版。原来的 UNIX 操作系

统是 1969 年由美国的贝尔实验室的 K.Thompson 和 D.M.Ritchie 开发成功的，是用汇编语言写的。

1972 年至 1973 年间，贝尔实验室的 D.M.Ritchie 在 B 语言的基础上设计出了 C 语言(取 BCPL 的第二个字母)。C 语言既保持了 BCPL 和 B 语言的优点(精练，接近硬件)，又克服了它们的缺点(过于简单，数据无类型等)。最初的 C 语言只是为描述和实现 UNIX 操作系统提供一种工作语言而设计的。1973 年，K.Thompson 和 D.M.Ritchie 两人合作把 UNIX 的 90%以上用 C 改写，即 UNIX 第 5 版。原来的 UNIX 操作系统是 1969 年由美国的贝尔实验室的 K.Thompson 和 D.M.Ritchie 开发成功的，是用汇编语言写的。

后来，C 语言多次做了改进，但主要还是在贝尔实验室内部使用。直到 1975 年 UNIX 第 6 版公布后，C 语言的突出优点才引起人们的普遍注意。1977 年出现了不依赖于具体机器的 C 语言编译文本《可移植 C 语言编译程序》，使 C 移植到其他机器时所需做的工作大大简化了，这也推动了 UNIX 操作系统迅速地在各种机器上实现。例如 VAX、AT&T 等计算机系统都相继开发了 UNIX。随着 UNIX 的日益广泛使用，C 语言也迅速得到推广。C 语言和 UNIX 可以说是一对孪生兄弟，在发展过程中相辅相成。1978 年以后，C 语言已先后移植到大、中、小、微型机上，已独立于 UNIX 和 PDP 了。现在 C 语言已风靡全世界，成为世界上应用最广泛的几种计算机语言之一。以 1978 年发表的 UNIX 第 7 版中的 C 编译程序为基础，BrianW.Kernighan 和 DennisM.Ritchie(合称 K&R)合著了影响深远的名著《The C Programming Language》，这本书中介绍的 C 语言成为后来广泛使用的 C 语言版本的基础，它被称为标准 C。1983 年，美国国家标准化协会(ANSI)根据 C 语言问世以来各种版本对 C 的发展和扩充，制定了新的标准，称为 ANSIC。ANSIC

比原来的标准 C 有了很大的发展。K&R 在 1988 年修改了他们的经典著作《The C Programming Language》，按照 ANSIC 标准重新写了该书。1987 年，ANSI 又公布了新标准——87ANSIC。

1990 年，国际标准化组织 ISO 接受 87ANSIC 为 ISOC 的标准(ISO9899—1990)。目前流行的 C 编译系统都是以它为基础的。本书的叙述基本上以 ANSIC 为基础。目前广泛流行的各种版本 C 语言编译系统虽然基本部分是相同的，但也有一些不同。在微型机上使用的有 Microsoft C、Turbo C、Quick C、BORLAND C 等，它们的不同版本又略有差异。

1.1.2 C 语言特点

C 语言对操作系统和系统使用程序以及需要对硬件进行操作的场合，用 C 语言明显优于其它高级语言，许多大型应用软件都是用 C 语言编写的。C 语言具有绘图能力强，可移植性，并具备很强的数据处理能力，因此适于编写系统软件。

（1）简洁紧凑、灵活方便

C 语言一共只有 32 个关键字,9 种控制语句，程序书写自由，主要用小写字母表示。它把高级语言的基本结构和语句与低级语言的实用性结合起来。 C 语言可以象汇编语言一样对位、字节和地址进行操作, 而这三 者是计算机最基本的工作单元。

（2）运算符丰富

C 的运算符包含的范围很广泛，共有种 34 个运算符。C 语言把括号、赋值、强制类型转换等都作为运算符处理。从而使 C 的运算类型极其丰富表达式类型多样化，灵活使用各种运算符可以实现在其它高级语言中难以实现的运算。

（3）数据结构丰富

C 的数据类型有：整型、实型、字符型、数组类型、指针类型、结构体类型、共用体类型等。能用来实现各种复杂的数据类型的运算。并引入了指针概念,使程序效率更高。另外 C 语言具有强大的图形功能, 支持多种显示器和驱动器。且计算功能、逻辑判断功能强大。

（4）C 是结构式语言

结构式语言的显著特点是代码及数据的分隔化,即程序的各个部分除了必要的信息交流外彼此独立。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。C 语言是以函数形式提供给用户的,这些函数可方便的调用,并具有多种循环、条件语句控制程序流向,从而使程序完全结构化。

（5）C 语法限制不太严格、程序设计自由度大

一般的高级语言语法检查比较严，能够检查出几乎所有的语法错误。而 C 语言允许程序编写者有较大的自由度。

（6）C 语言允许直接访问物理地址，可以直接对硬件进行操作

因此既具有高级语言的功能，又具有低级语言的许多功能，能够象汇编语言一样对位、字节和地址进行操作,而这三者是计算机最基本的工作单元，可以用来写系统软件。

（7）C 语言程序生成代码质量高，程序执行效率高

一般只比汇编程序生成的目标代码效率低 10 ～ 20%。

（8）C 语言适用范围大，可移植性好

C语言有一个突出的优点就是适合于多种操作系统, 如DOS、UNIX,也适用于多种机型。

1.1.3 简单的 C 程序

为了使用 C 语言编写程序，必须了解 C 语言，并且能熟练地使用 C 语言。本教程以以下案例引入 C 语言程序编写规则。

```
#include <stdio.h>                    //这是编译预处理指令
#include <stdlib.h>                   //这是编译预处理指令
void main()                           //定义主函数
{                                     //函数开始标记
    printf("Hello World \n");         //输出指定行信息
system("pause");                      //程序暂停，敲下任意键继续
}                                     //函数结束标记
```

运行结果：

Hello World

上面的 Hello world 就是一个最简单的完整程序，只有 7 行。一个 C 语言是由两个部分组成：定义和函数，至少有一个主函数。前两行#include <stdio.h>和#include <stdlib.h>可以把它看做定义部分。第 3 行到第 7 行是主函数部分。

前两行#include <stdio.h>和#include <stdlib.h>意为包含文件，也是引用的意思，引用的文件中包含有诸多库函数，可以在后期的程序中直接使用。

程序的第 3 行是主函数的开始。main 这个函数是 C 语言的重要内容，如果我们编写的是一个可执行程序，运行这个程序必须有一个唯一的入口，C 语言的这个入口就是 main。

程序中双斜杠//表示注释，可以在注释中添加程序说明。

任务 1.2 Visual studio2010 软件的使用

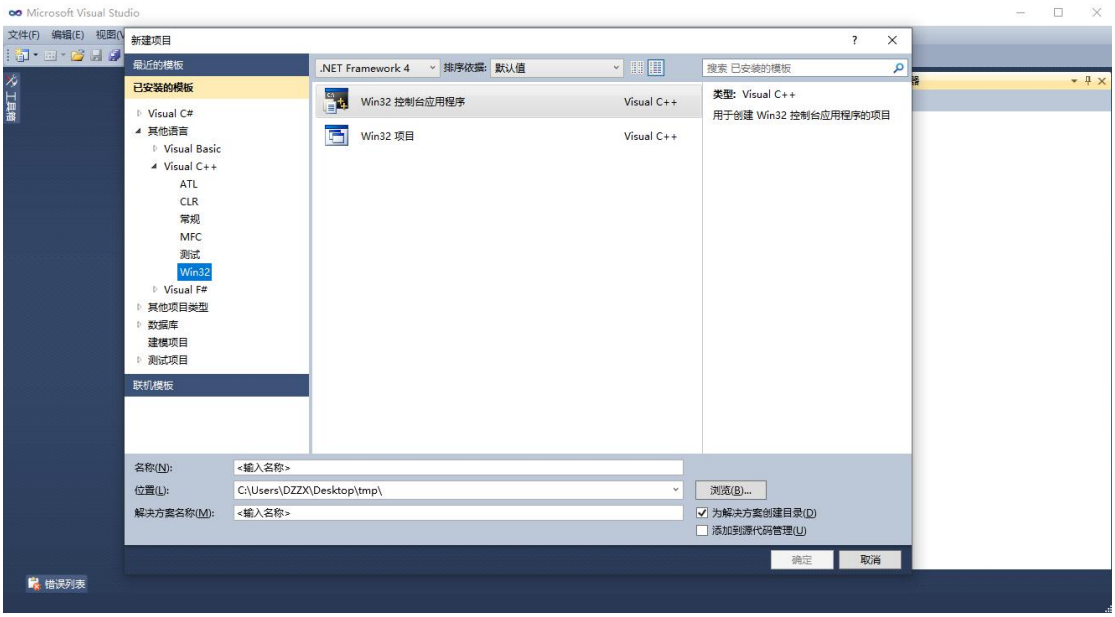
任务目标

掌握用 Visual studio2010 开发 C 语言程序的方法和步骤

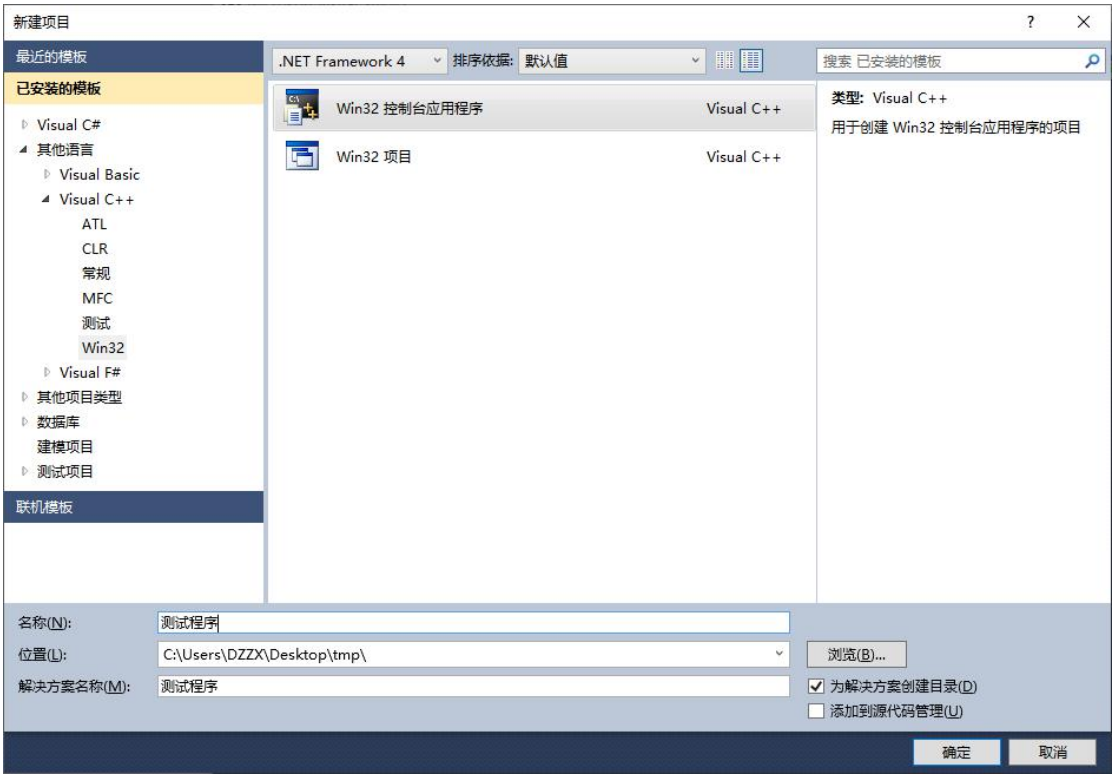
任务实施

下面介绍当前主流软件 Visual studio2010 下，开发 C 语言程序的操作步骤：

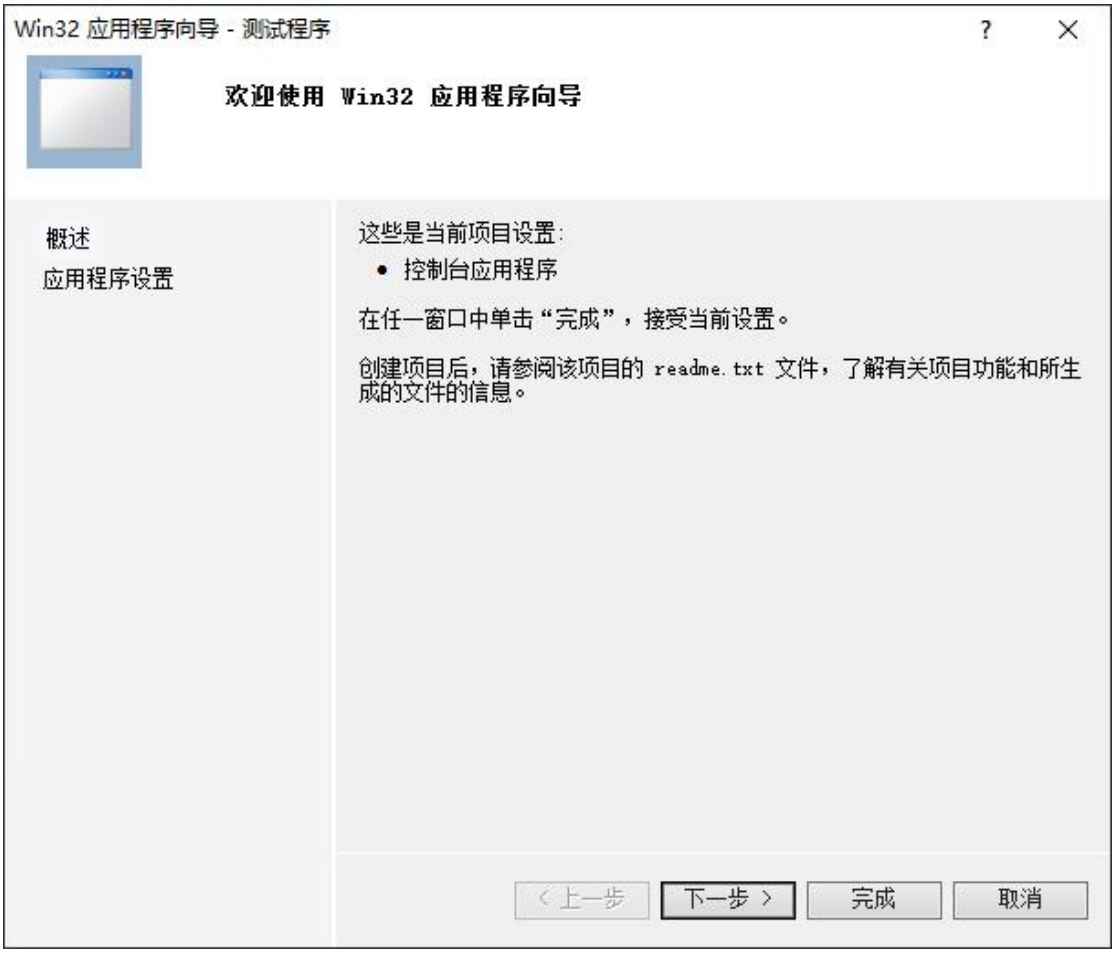
（1）点击“文件 -> 新建 -> 项目”



（2）依次选择“其他语言 -> Visual C++ -> Win32”点击“Win32 控制台应用程序”，输入项目名称、项目位置，点击“确定”

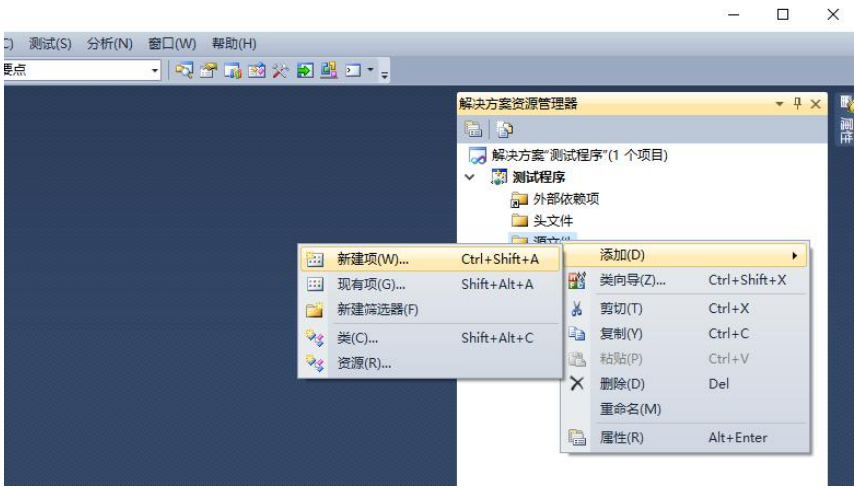


（3）在弹出的对话框中点击“下一步”

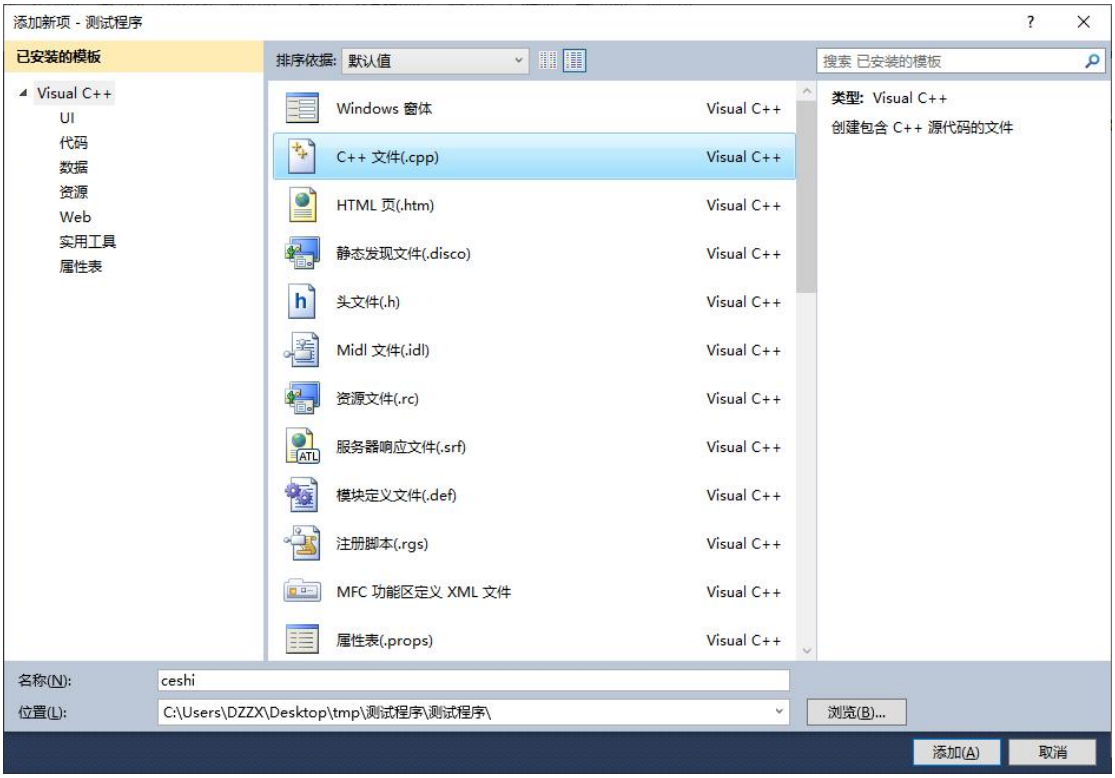


（4）在弹出的对话框中按照以下图样设置参数，点击“完成”

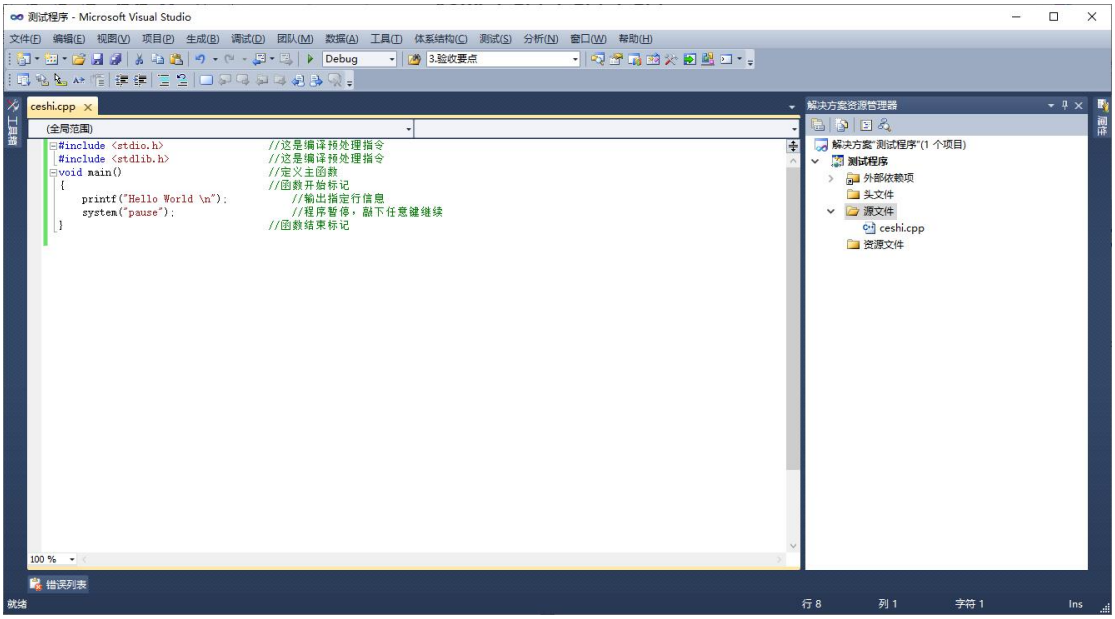
（5）右击“解决方案资源管理器 -> 源文件”选择“添加 -> 新建项”



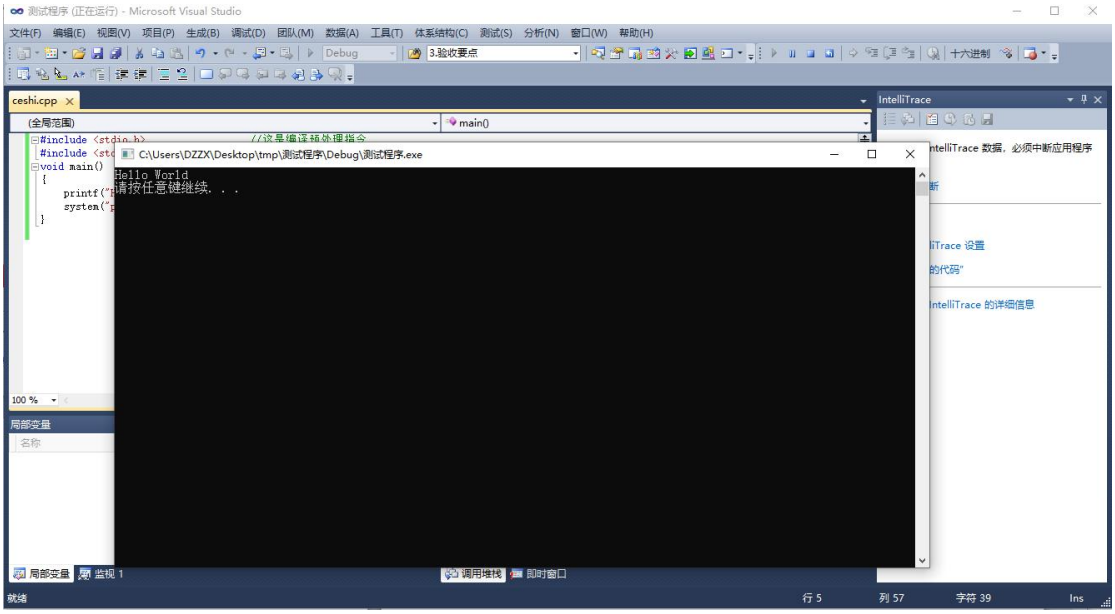
（6）在弹出的对话框中选择“C++文件(.cpp)”，输入名称后，点击“添加”按钮。



（7）输入案例程序



（8）点击菜单“调试 -> 启动调试”，查看运行结果



任务 1.3 常见错误分析

任务目标

了解C程序中常见的错误，掌握检查程序的方法

任务实施

在编写程序的过程中，难免会出现代码错误情况，编写程序的时候，一定要尽量减少错误的代码。

1.代码的错误包括逻辑错误和语法错误。

逻辑错误的代码的算法出现错误，代码不能完成对应的功能。这种错误，能编译成功，但不能得到正确的运行结果。

语法错误是指程序代码不符合C语言的编写规则，程序不能编译通过。

2.常见语法错误

（1）分号丢失

分号是C语言程序语句的重要组成部分，每条语句都必须以分号结尾。

（2）括号不成对出现

语句体中的大括号、函数使用的小括号，运算符中的小括号等，必须成对出现。

（3）语句中出现中文字字符

C 语言的语句中只能识别英文字符，中文字符无法编译。特别是 C 语言中的各种英文符号，一定要输入英文符号。

（4）C 语言要区分大小写字母

在写代码时，一定要注意区分大小写。例如在定义变量的时候，定义了小写的变量，使用大写的变量将会认为变量不存在。

项目小结

本任务主要讲解了 C 的语言的发展和基本结构；用 viaual studio 2010 如何调试 C 语言程序。通过本项目的学习，大家要掌握如何调试 C 程序，能快速、准确判断出 C 程序中的错误代码，为后面的项目学习和程序调试打下基础。

初学 C 语言程序时，应注意以下内容：

（1）C 语言的语句中只识别英语字符（不包括提示和注释），因此，在输入标点时，应特别注意。

（2）每条语句的末尾必须以分号结束。

（3）C 语言要区分大小写。

（4）括号和引号要成对出现，否则会编译出错。

（5）变量要先定义后使用

习题

一、选择题

1.以下叙述不正确的是（　　）

A. 一个 C 源程序可以由一个或多个函数构成。

B. 一个 C 源程序必须包含一个 main 函数。

C. 在对一个 C 程序进行编译过程中，可以发现注释中的拼写错误。

D. C 程序的基本组成单位是函数。

2. 在 C 程序中，main（）函数的位置是（　　）

A. 必须作为第一个函数　　B. 必须作为最后一个函数

C. 可以在任意位置　　D. 必须放在它所调用的函数之后

3. 计算机能直接执行的程序是（　　）

A. 源程序　　B. 目标程序　　C. 汇编程序　　D. 可执行程序

4. 以下说法中正确的是（　　）

A. C 源程序可以直接运行产生结果

B. C 源程序经编译后才可直接运行产生结果

C. C 源程序经连接后才可直接运行产生结果

D. C 源程序经编译和连接后才可直接运行产生结果。

二、填空题

1. C 语言源程序文件的后缀是_____，经过编译后生成文件的后缀名是_____ ，经过连接生成文件的后缀是_____。

2. C 语言源程序中，块注释部分以_____开始，并且以_____结束。

三、编程

1. 模仿编写程序，输出以下信息：

******这是第一个 C 程序！*****

2. 给每行加上注释。

项目二 基本数据类型、运算符和表达式

一个程序的主体是由语句组成的，语句决定了如何对数据进行处理，而在程序中数据和数据处理的表示与高级语言中的数据类型、运算符与表达式有关。变量和常量是程序处理的两种基本数据。声明语句时要说明变量的名字及类型，数据的类型决定该数据可取值的范围以及可以对该数据进行的操作。运算符指定将要进行的操作。表达式则把变量与常量组合起来生成新的值。本章将详细讲述这些内容。

任务 2.1　C 语言的数据类型

任务目标

了解 C 语言中的基本数据类型

任务实施

C 语言中常见的三种数据的分别是整型、浮点型、字符型。

2.1.1 整型

整型是用来表示整数的，主要用符号 int 表示。

例如：

int a=3 ;　　//定义一个整数类型变量 a，a 的值是 3

为了满足不同的需要，整型按照可表示的范围分为以下几类：

1、整型（又叫一般整型）：使用 int 关键字表示;

2、短整型：使用 short int 关键字表示，简写为 short;

3、长整型：使用 long int 关键字表示，简写为 long;

短整型用来表示较小范围的数，长整型用来表示较大范围的数。

2.1.2 浮点型

浮点型在 C 语言中主要用来表示小数，分为单精度浮点（float）类型和双精度浮点（double）类型。双精度的表示精度会更细点，并且表示的范围也会更大。

注意：给 float 类型赋值的时候需要以 f 结尾，否则会有警告。

例如：float f=3.14f;　//定义一个单精度类型变量 f，f 的值是 3.14

double d=3.14;　//定义一个双精度类型变量 d，d 的值是 3.14

2.1.3 字符型

1. 字符常量

括在一对单引号中的一个字符，如：‘A’、‘x’、 ‘D’、‘3’、 ‘X’ 等都

是字符常量。

注意：

单引号只是字符常量的定界符，而非字符常量本身内容；

大小写字母是有区别的，如‘x’和‘X’是两个不同的字符常量；

一个字符常量占 1 个字节，存放的是字符的 ASCII 码值。

2. 字符变量

用来存放字符常量，只能放一个字符。

例：char c=‘a’; //定义一个字符变量 c，c 的值是字符常量 a

也可以用字符对应的 ASCII 码赋值，如下:

char c=97;

一个字符变量在内存中也占一个字节。

3. 转义字符：

在 C 语言中，有一类特殊字符，是以字符\开头的字符序列，无法用一般形式表示，只能采用这种特殊的形式表示，这类字符被称为转义字符。

例’\n’，它代表一个“换行”符。

任务 2.2　常量与变量

任务目标

了解各种基本数据类型的常量和变量

任务实施

2.2.1 常量

在程序运行过程中，其值不能被改变的量。

1. 直接常量(字面常量)：

整型常量：如 12、0、-3 等

实型常量：如 4.5、-1.234 等

字符常量：如‘a’、‘1’等，用单引号表示；

字符串常量：如“a”、“abc”、“1”，用双引号表示。

2. 符号常量：

符号常量是用一个标识符来代替一个常量，常借助于预处理命令 #define 来实现。

定义形式：#define　标识符　字符串

如：#define　　PI　　3.1415926535

注意：

①习惯上，符号常量用大写字母表示；

②定义符号常量时，不能以“；”结束；

③一个#define 占一行，且要从第一列开始书写；

④一个源程序文件中可含有若干个 define 命令，不同的 define 命令中指定的“标识符”不能相同；

2.2.2 变量

在程序运行过程中，其值会发生变化。

变量的声明和定义是一体的。其语法形式如下:

变量类型　变量名

例如：float radius　　//定义一个浮点型变量，变量名为 radius

注意：

(1) 每个变量必须有一个名字，变量名是标识符（用来标识数据对象，是一个数据对象的名字）。

(2) 命名规则是以字母或下划线开始，后跟字符、数字或下划线。

例：x1，_average，lotus_1_2_3，#abc，1fs，M.D.Jhon

(3) 变量必须先定义再使用

(4) 变量名不能是关键字(即保留字，是 C 编译程序中保留使用的标识符。 如：auto、break、char、do、else、if、int 等)

C 语言中的关键字如表 2.1 所示。

表 2.1 关键字

auto	break	case	char	const	continue	default
do	double	else	emun	extrn	float	for
goto	if	int	long	register	return	short
signed	sizeof	static	struct	switch	typedef	union
unsigede	void	volatile	while			

任务 2.3 赋值语句

任务目标

了解 C 语言中的基本数据类型

任务实施

C 语言中的变量用来表示在计算过程中一直变化的数据。计算机程序是要使用这些变化的数据来计算所需的结果的，赋值运算的作用就是把变化的数据交给变量保存起来。

2.3.1 赋值运算符

C 语言中使用等号“=”作为赋值运算符，作用是将一个值(变量值、表达式计算结果等)复制给另一个变量。赋值运算符常见形式：

变量 = 值/变量值/表达式;

例如：

```
int a=1 ; //声明一个整型的变量 a，赋值为 1
a=1+1; //先计算表达式 1+1，然后将结果值赋给变量 a
a=b;     //将变量 b 的值赋值给变量 a
b=a-10;    //先计算 a-10，然后将计算结果赋值给变量 b
```

c=a+b; //先计算 a+b，然后将计算结果赋值给变量 c

注意:

20=30; //错误写法

100=a; //错误写法

a+2=5; //错误写法,左边不能是表达式

2.3.2 赋值表达式

将赋值运算符、小括号、操作数连接起来、符合 C 语言规则的式子被称为赋值表达式，参与运算的操作数可以是值、变量、表达式、函数……等。

例如：a= 1+1、a=2、a=a+1、b=a+1 等。

b=a+1 的读法是“a+1 的值赋值给 b”， a=a+1 的读法是“a+1 的值赋值给 a”

2.3.3 复合赋值运算符

在赋值运算符前加上其他运算符，就可以构成复合运算符。例如：在“=”前加一个“-”就构成复合运算符“-=”。在 C 语言中，常用的复合赋值运算有以下几种：

a+=10; //等价于 a=a+10;

a-=10; //等价于 a=a-10;

a*=10; //等价于 a=a*10;

a/=10; //等价于 a=a/10;

任务 2.4 输出数据

任务目标

了解函数 Putchar( )和 Printf( )的使用方法

任务实施

2.4.1 putchar()

Putchar()是字符输出函数，功能是在终端（显示器）输出单个字符。其格式为 putchar(c)，其中 c 可以是被单引号（英文状态下）引起来的一个字符，可以是介于 0~127 之间的一个十进制整型数，也可以是事先用 char 定义好的一个字符型变量。

如代码:

```
putchar('A');        //输出大写字母 A
putchar(x);          //输出字符变量 x 的值
putchar('\n');       //换行
```

注意：

（1）putchar 函数只能用于单个字符的输出，且一次只能输出一个字符。

（2）在程序中使用 putchar 函数，在程序（或文件）的开头加上编译预处理命令,即：#include < stdio.h>。

例如：

```
#include < stdio.h >        //预处理指令
 void main()       //    main()主函数,一个 C 语言必须有一个主函数
 {
    char ch1='N', ch2='E', ch3='W';
    putchar(ch1); putchar(ch2); putchar(ch3);
    putchar('\n');
    putchar(ch1); putchar('\n');
    putchar('E'); putchar('\n');
    putchar(ch3); putchar('\n');
 }
```

程序运行结果如下：

NEW

N

E

W

2.4.2 printf()

printf()是格式输出函数，功能是按照用户指定的格式，把指定的数据输出到屏幕上。printf 函数的格式为:

printf(“格式控制字符串”,输出表项);

格式控制字符串用来说明输出表项中各输出项的输出格式;

输出表项列出了要输出的项，各输出项之间用逗号分开。输出表项也可以没有，则表示输出的是格式字符串本身。

格式控制字符串有两种：格式字符串和非格式字符串。非格式字符串在输出的时候原样打印;格式字符串是以%打头的字符串,在”%”后面跟不同格式字符，用来说明输出数据的类型、形式、长度、小数位数等。

表 2. 2 常用的输出格式及含义

格式字符	含　义
%d , %i	按十进制整型数据的实际长度输出。
o%	以八进制形式输出无符号整数
x%	以十六进制形式输出无符号整数
%f	以小数形式输出实数, 可以指定精度，例如%.2f 就是保留 2 位小数。
C%	输出单个字符
S%	输出字符串

例如：

```
# include <stdio.h>
int main(void)
{
    int i=10;
    int j=3;
    printf("i=%d, j=%d\n", i, j);
    return 0;              //表示 main 函数执行到此处时结束
}
```

程序运行结果如下：

```
i = 10, j = 3
```

任务 2.5　输入数据

任务目标

了解函数 getchar()和 scanf ( )的使用方法

任务实施

2.5.1 getchar()

getchar()输入数据函数,功能是接收用户从键盘上输入的一个字符。其一般调用形式为：

```
getchar();
```

getchar 会以返回值的形式返回接收到的字符,用法如下:

```
char c;            //定义字符变量 c
c=getchar();     //将读取的字符赋值给字符变量 c
```

例如：

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

```
{
    char    ch;
    printf("please input two character: ");
    ch=getchar();
    putchar(ch);putchar('\n');
    putchar(getchar());
    putchar('\n');
}
```

程序运行情况如下：

please input two characters: ab↙（若此时输入 ab，并回车）

a

b

注意：getchar 函数只能用于单个字符的输入，一次输入一个字符。程序的功能是输入一个字符，显示一个字符，回车换行，再输入并显示一个字符。而运行时字符是连续输入的，运行结果却是正确的，这是因为输入字符后，它们暂存于键盘的缓冲区中，然后由 getchar 函数从键盘缓冲区中一个一个的取出来。

2.5.2 scanf()

scanf 函数称为格式输入函数，scanf 函数的功能与 printf 函数正好相反，即按照格式字符串的格式，从键盘上把数据输入到指定的变量之中。Scanf 函数的调用的一般形式为：

scanf(“格式控制字符串”，输入项地址列表);

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

例如：

```
#include <stdio.h>
int main(void) {} {
{   int i;
scanf("%d", &i);     //&i 表示变量 i 的地址，&是取地址符
printf("i=%d\n",i);
        return 0;
}
```

运行上面程序时，从键盘上输入字符 123，然后%d 将这三个字符转换成十进制数 123，最后通过“取地址”找到变量 i 的地址，再将数字 123 放到以变量 i 的地址为地址的变量中，即变量 i 中，所以最终的输出结果就是 i=123 。

如果要一次给多个变量赋值，比如给两个变量赋值就写两个%d ，然后“输入项地址列表”中对应写上两个“取地址变量”;给三个变量赋值就写三个%d，然后“输入项地址列表”中对应写上三个“取地址变量”.....

例如：

```
#include <stdio.h>
int main(void) {} {
{   int i，j;
scanf("%d%d", &i， &j);
printf("i=%d，j=%d\n"，i，j);
        return 0;
}
```

Scanf 函数时注意事项：

1、在 scanf 的“输入项地址列表”中，变量前面的取地址符&不要忘记。

2、scanf 中双引号内，除了“格式控制字符串”外什么都不要写。

3、“格式控制字符串”和“输入项地址列表”无论在“顺序上”还是在“个数上”一定要一一对应。

任务 2.6　运算符和表达式

任务目标

1、认识掌握算术运算符及表达式

2、自增、自减运算符的使用

3、sizeof 运算符的使用

4、三目运算符的使用

5、逗号运算符的使用

任务实施

运算符是一种告诉编译器执行特定的数学或逻辑操作的符号。C 语言内置了丰富的运算符类型。

2.6.1 算术运算符

算术运算符是用于执行程序中的数学运算，C 语言中常用的算术运算符有以下 5 种：

①加法或者正值运算符“+”。作为加法运算符时是双目运算符，也就是需要有 2 个操作数参与运算，例如：a+b、1+3 等。

②减法或者负值运算符“-”。

③乘法运算符“*”。

④除法运算符“/”。

⑤求余运算符“%”，属于双目运算符，要求参与运算的两个操作数都必须是整数，求余运算的结果是两个数相除的余数，示例代码如下：

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

```
int main()
{
int a=666；
int b=20;
printf("%d\n",b%a);
getchar();
return 0;
}
```

2.6.2 算术表达式

将算术运算符、小括号、操作数连接起来、符合 C 语言规则的式子被称为算术表达式， 参与运算的操作数可以是字面量、变量、函数等。

例如：a+1、a-b、1*10、20/2、30%4…..等等。

2.6.3 算术运算符优先级

在 C 语言中，当算术表达式由多个不同的算术运算符组成时，会按照运算符的优先级进行运算：先乘除后加减、先括号里再括号外，优先级相同，按照自左向右的顺序进行运算。

例如：a*10+(100%3)-b/10

小括号优先级最高，先计算(100%3)。*与/优先级相同，自左向右运算，先计算 a*10， 再计算 b/10。+与-优先级相同，自左向右运算，先计算加法，再计算减法。

2.6.4 自增(++)运算符

自增运算符 “++”由 2 个“+”组成，作用是将变量值增加 1，在 C 语言中，“++”的位置不同表达的意思也不相同。

++x; //x 先自增 1，然后再参加运算

x++; //x 先参与运算，然后自增 1

2.6.5 自增运算表达式

将自增运算符、小括号、操作数连接起来、符合 C 语言规则的式子被称为自增运算表达式，参与运算的操作数只能是变量，不能是字面量、表达式等。

例如：

```
++100        //错误写法，不能是字面量
(a+1)++      //错误写法，不能是表达式
```

前面讲过，“++”的位置不同表达的意思也不相同。

如下面实例：

```
#include<stdio.h> int main()
{
    int a=1; int b=0;
    b=a++;    //后++
    printf("a=%d\n",a);
    printf("b=%d\n",b); getchar();
    return 0;
}
```

运行结果：

```
a=2
b=1
```

```
#include<stdio.h> int main()
{
    int a=1; int b=0;
    b=++a;    //前++
    printf("a=%d\n",a);
```

```
    printf("b=%d\n",b); getchar();
    return 0;
}
```

运行结果：

```
a=2
b=2
```

分析：如果是“变量++”，返回变量自加之前的值；如果是“++变量”，返回变量自加之后的值。因此，上述程序中，b=a++和 b=++a 执行过程等价于以下形式：

b=a++等价于	**b=++a** 等价于
b=a; **a=a+1;**	**a=a+1;** **b=a;**

2.6.6 自减(--)运算符

自减运算符“--”由 2 个“-”组成，作用是将变量值减少 1。也分为 i--、--i 的区别。和自加(++)运算符使用方法类似。

四、sizeof 长度运算符

用于计算变量、字面量、类型所占字节数。

sizeof(类型)、sizeof(字面量)、sizeof(变量)

上述表达式的运算结果为：括号中的类型在当前系统下所占字节数。

注意：

(1)不同类型的数据的大小在不同的平台下有所区别，但是 c 标准规定所有编译平台都应该保证 sizeof(char) 等于 1。

例如：sizeof(int)

在 16 位平台 int 下是 2；在 32 位平台 int 下是 4；在 64 位平台下 int 是 8。

(2)sizeof 是运算符，不是函数。

(3)当表达式作为 sizeof 的操作数时，它返回表达式的计算结果的类型大小，不对表达式求值。

2.6.7 三目运算符（条件运算符）

格式：表达式 1？表达式 2：表达式 3

求值规则为：如果表达式 1 的值为真，则以表达式 2 的值作为整个条件表达式的值，否则以表达式 3 的值作为整个条件表达式的值。

例如：max=（a>b）?a:b,执行结果就是将 a 和 b 中大者赋给 max。

实例 :输出 a，b 中较大者

```
#include<stdio.h>
int main (void)
{
int a,b;
scanf("%d%d",&a,&b);
printf("%d\n",a>b?a:b);
return 0;
}
```

使用条件表达式时，还应注意以下几点：

(1)条件运算符的优先级低于关系运算符和算术运算符，但高于赋值符；

(2)条件运算符?和：是一对运算符，不能分开单独使用；

(3)条件运算符的结合方向是自右至左。

2.6.8 逗号运算符

逗号运算符是双目运算符，逗号运算符确保操作数被顺序地处理：

先计算左边的操作数，再计算右边的操作数。右操作数的类型和值作为整个表达式的结果。

格式: 表达式 1，表达式 2

例如：a=2*6,a-4,a+15;

逗号运算符的运算顺序是自左向右的，因此上述赋值语句的求值顺序为：先计算 2*6 并赋予 a（结果是 a=12），再计算 a-4（只计算，不赋值），最后计算 a+15（只计算，不赋值），最终以 27 作为整个逗号表达式的值。

注意：

(1)逗号运算符的优先级是所有运算符中最低的；

(2)逗号运算符允许将多个表达式组合成为一个表达式

运算符优先级表

优先级	运算符	名称或含义	表达式形式	结合方向	说明
1	[ ]	数组下标	数组名[常量]	左到右	单目运算符
	()	圆括号	(表达式)/函数名		单目运算符
	.	成员选择(对象)	对象.成员名		单目运算符
2	-	负号预算符	-操作数	右到左	单目运算符
	++	自增运算符	++变量名/变量名++		单目运算符
	--	自减运算符	--变量名/变量名--		单目运算符
	!	逻辑非运算符	!操作数		单目运算符
	sizeof	长度运算符	~操作数		
3	/	除	操作数/操作数	左到右	双目运算符
	*	乘	操作数*操作数		双目运算符

	%	余数(取模)	整型操作数/整型操作数		双目运算符
4	+	加	操作数+操作数	左到右	双目运算符
	-	减	操作数-操作数		双目运算符
5	>	大于	操作数>操作数	左到右	双目运算符
	>=	大于等于	操作数>=操作数		双目运算符
	<	小于	操作数<操作数		双目运算符
	<=	小于等于	操作数<=操作数		双目运算符
6	==	等于	操作数==操作数	左到右	双目运算符
	!=	不等于	操作数!=操作数		双目运算符
7	&&	逻辑与	操作数&&操作数	左到右	双目运算符
8	||	逻辑或	操作数||操作数	左到右	双目运算符
9	?:	条件运算符	操作数 1?操作数 2:操作数 3	左到右	三目运算符
10	=	赋值运算符	变量=操作数	左到右	双目运算符
	/=	除后赋值	变量/=操作数		双目运算符
	*=	乘后赋值	变量*=操作数		双目运算符
	%=	取模后赋值	变量%=操作数		双目运算符
	+=	加后赋值	变量+=操作数		双目运算符
	-=	减后赋值	变量-=操作数		双目运算符
11	,	逗号运算符	操作数，操作数……	左到右	从左到右顺序

项目小结

本项目主要讲解了C语言中的基本数据类型、变量和常量，运算符和表达式，常用的输入输出函数。本项目知识点多，需要理解和掌握的内容比较繁杂，同时，又是编写程序的基础，只有牢牢掌握本项目相关知识，才能更好的学习后续项目的内容。

习题

一、写出以下程序的运行结果并上机验证。

(1)以下程序的运行结果为_____

```
#inclue <stdio.h>
int main（）
{
   int a = 0 , b = 6 ;
   printf（"%d ,  %d \n " , ++a ,b --）;
   return   0 ;
}
```

（2）以下程序的运行结果为_____

```
#include <stdio.h>
int main（）
{
  int a = 135 , b , c ;
  b = a / 100 ;
  c = b % 2 ;
  printf（"%d ,%d\n",b,c）;
}
```

二、编写程序

1.编写程序，实现将 a、b 两个整数的值交换并输出。

2. 编写一个程序，输入半径，输出圆周长、圆面积及圆体积。

项目三 简单程序设计

著名计算机科学家 Nikiklaus Wirth 指出：程序 = 数据结构 + 算法。数据结构是对数据的描述，指在程序中要用到哪些数据，以及这些数据的类型和数据的组织形式；算法是指对数据进行什么样的操作。数据是操作的对象，操作的目的是对数据进行处理，以便得到预期的结果。

实际上，一个程序除了以上这个重要的要素外，还应当采用程序适当的程序设计方法来进行程序设计，另外，还要选择某一种计算机语言来表示。本章先介绍程序设计中的初步知识，为后面各章的学习打下基础。

任务 3.1　C 语句概述

任务目标

了解 C 语言语句的基本类型和特点

任务实施

为实现一个特定功能而编制的程序包含若干条语句。这些语句有可能是控制语句，也可能是函数调用语句，那么，C 语言中，语句包括哪些呢?

C 语言中，语句分为以下五类：

3.1.1 控制语句

完成控制功能的语句，在 C 语言中，控制语句只有 9 种，分别是：

If … else…	条件语句
switch	多分支语句
while	循环语句
do … while	循环语句
for	循环语句
continue	结束本次循环语句
break	中止循环或退出 switch 语句
retrun	返回语句
goto	跳转语句

3.1.2 函数调用语句

由函数调用加分号构成的语句，如：

printf(“Hello, The　C programming　Language”);

3.1.3 表达式语句

由表达式构成的语句，它是在表达式后面加上分号，如：

num=1　　//这是一个表达式，而

num=1;　　//这就是一个语句。

3.1.4 复合语句

用一对{ }括起来的语句，称为复合语句，它可以把一些语句看成是一个整体。通常出现在分支程序或循环程序中。如：

```
{ total = a1 + a2;
  ave = total / 2;
  printf(“ %f ” , ave ) ; }
```

注意:复合语句最后一句的分号必须加上，否则，程序调试会报错。

5. 空语句

即只有一个分号，其余的什么都没有。如下面的这一条语句就是空语句：

```
;
```

任务 3.2 程序的基本结构

任务目标

了解程序的三种基本结构。

任务实施

程序的基本结构有三种：

3.2.1 顺序结构

见图 3.1。一般情况下，程序总是按照代码的顺序，从上向下，依次执行。即先执行语句 A，再执行语句 B，再执行语句 C。

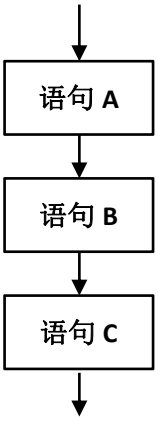


图 3.1 顺序结构

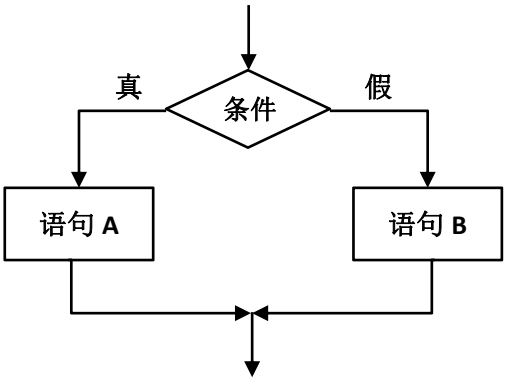


图 3.2 选择结构

3.2.2 选择结构

见图 3.2。先判断条件是否成立，如果成立（或称为“真”），就执行语句 A，否则语句 B。注意，语句 A 和语句 B 不能同时执行。不管是执行语句 A 或语句 B，执行完后，汇合到一个出口执行下一条语句。

3 .2.3 循环结构

循环结构有两种类型，先判断后循环和先循环后判断，前则称为“当型循环”，后则称为“直到型循环”。

（1）“当型循环”

见图 3 - 3，先判断条件是否成立，如果条件成立（为“真”），则执行语句 A，然后跳转到条件继续判断，反复进行，当条件不成立时，退出循环。

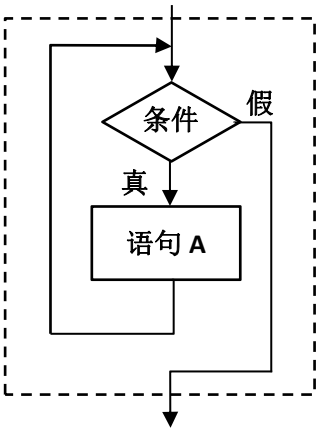


图 3- 3　“当型循环”

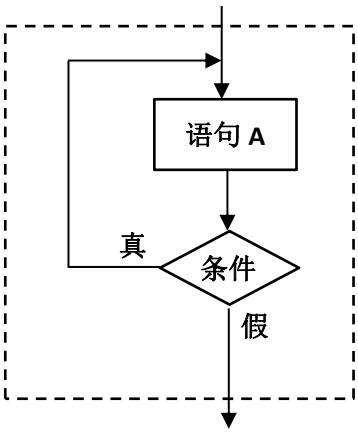


图 3-4　“直到型循环”

（2）“直到型循环”

见图 3 - 4，先执行语句 A，然后判断条件是否成立，如果条件成立（为“真”），则跳转回去继续执行语句 A，直到条件不成立（为“假”），则退出循环。

实践证明，由以上的基本结构组成的程序能处理任何复杂的问题。上面的结构图中的语句 A、语句 B、语句 C 等，可以是简单的一个语句，也可以是复合语句，还可以是一个基本结构，即选择结构里也可以出现选择或循环结构，循环结构里也可以出现选择或循环结构。

任务 3.3　算 法

任务目标

1.了解算法的概念

2.了解算法的性质

任务实施

计算机算法是以一步接一步的方式来详细描述计算机如何将输入转化为所要求的输出的过程，或者说，算法是对计算机上执行的计算过程的具体描述。

一个算法必须具备以下性质:

(1)算法必须是正确的，即对于任意的一组输入，包括合理的输入与不合理的输入，总能得到预期的输出。如果一个算法只是对合理的输入才能得到预期的输出，而在异常情况下却无法预料输出的结果，那么它就不是正确的。

(2)算法必须是由一系列具体步骤组成的，并且每一步都能够被计算机所理解和执行，而不是抽象和模糊的概念。

(3)算法的每个步骤都有确定的执行顺序，即上一步在哪里：下一步是什么，都必须明确，无二义性。

(4)算法无论有多么复杂，都必须在有限步之后结束并终止运行，即算法的步骤必须是有限的。在任何情况下，算法都不能陷入无限循环中。

一个问题的解决方案可能不止一种，也就是算法不是唯一的，通常选择方法简单，步骤较少的方法，所以，要有效的解决问题，不仅仅需要保证算法正确，还要考虑算法的质量。

任务 3.4　算法的表示

任务目标

1.了解算法的表示方法

2.会用流程图来描述算法

任务实施

常见的算法表示方法包括：自然语言，流程图，N-S 图，伪代码等。

3.4.1 自然语言表示算法

自然语言就是人们日常使用的语言，这种表示方法通俗易懂，但

文字冗长，易出现歧义，常常要根据上下文才能判断正确含义。

用自然语言不方便描述计算机中的算法，特别是包含了选择结构和循环结构的算法，因此，一般不用自然语言表示算法。

3.4.2 用流程图表示算法

流程图是用图框来表示各种操作，这种表示方法直观形象，易于理解，美国标准化协会 ANSI 规定了一些常用的流程图符号，见图 3 - 5，已经在世界各国广泛使用。

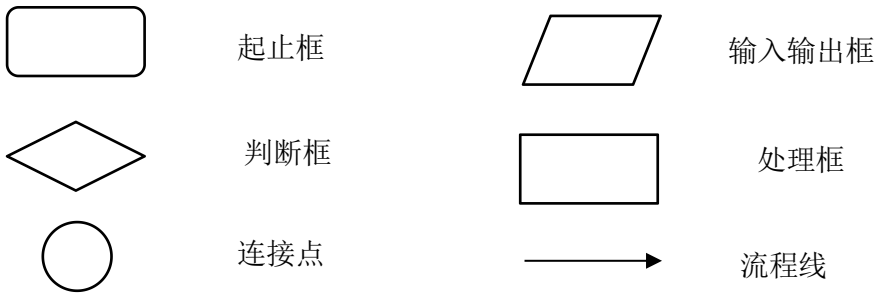


图 3 - 5　流程图符号

例 1：输入两个数值，计算者之和。流程图见图 3 - 6。

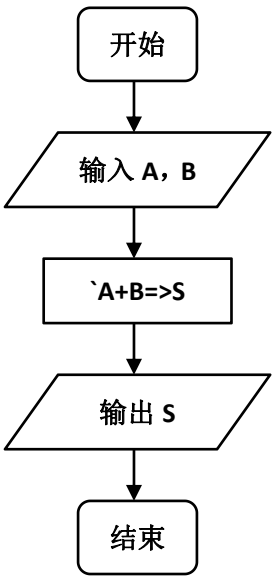


图 3 - 6　求两数和

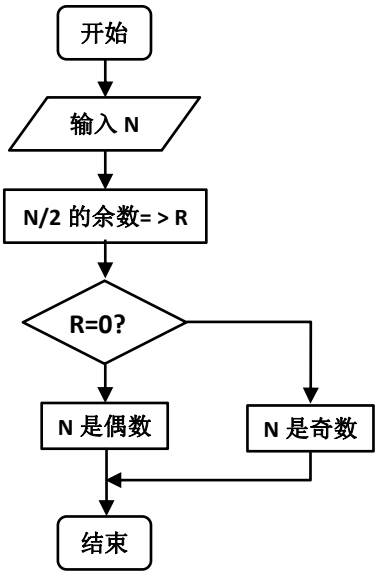


图 3 - 7　判断数的奇偶

例 2：输入一个数，判断其奇偶性。流程图见图 3.7

通过以上例子，可以看出流程图是表示算法比较好的工真。一个流程图应当包括以下几部分：

（1）表示相应操作的图框；

（2）带箭头的流程线；

（3）图框内必要的文字说明。

3.4.3 用 N-S 流程图表示算法

这种表示方法是在一个矩形框内表示一个完整的算法，去掉了流程线。

N-S 流程图表示如下：

图 3 - 8 顺序结构

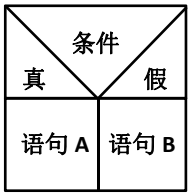


图 3 - 9 选择结构

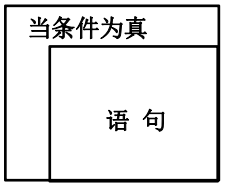


图 3- 10 当型循环结构

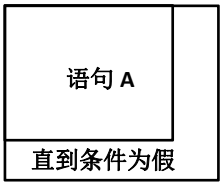


图 3- 11 直到型循环结构

任务 3.5 结构化程序设计方法

任务目标

了解结构化程序设计的有关方法

任务实施

前面已经介绍的三种程序结构，就是结构化的程序，便于编写、阅读、修改和维护。结构化程序设计强调设计风格和程序结构的规范性，提倡结构清晰。如果面临一个复杂的问题，难以一下子写出一个层次分明，算法正确的程序，则要把这个复杂的问题分解起逻辑清晰而几个部分来解决。

通常，采取“自顶向下，逐步清晰，模块化设计，结构化编码”的方法进行。

自顶向下，逐步清晰的设计方法，是将复杂问题由抽象到具体化的过程，将问题细化成若干个小问题，然后对每一个小问题再依次细化，直到不能细化为止。

当程序比较复杂时，通常采用模块化设计方法。根据程序模块的功能将它划分为若干个子模块，如果子模块规模还大，则可以继续划分为更小的模块，这个过程是通过自顶向下的原则进行。C 语言中的子模块是通过函数来实现的。

设计好结构化算法后，还要善于进行结构化编码。编码就是指将已设计好的算法用计算机语言来表示，即根据算法正确写出相应的计算机代码。结构化的编程语言（如 C 语言，Basic 等）都能进行编写相应的结构化代码。

学习程序设计，不是为了掌握某一门编程语言，而是要学习程序设计的方法，掌握编程思想，具体的编程语言只是为了实现程序功能，为不是最终目的。只有掌握了编程思想，掌握了算法，无论用哪种编程语言都可以实现，因此，不要拘泥于一种具体的编程语言。

项目小结

本项目是学习 C 语言程序设计的基础。介绍了 C 语言的语句、程序的基本结构以及算法的相关知识。

学习程序设计，是为了掌握程序设计的理念和方法，不是为了学习某一种具体编程语言。高级语言有很多种，每种语言也在不断发展中，所以不能只拘泥于一种具体语言，关键要掌握算法，只要掌握了算法，用任何语言进行程序设计都会得心应手。

习题

1.什么是算法?请从日常生活中出 3 个例子，描述它们的算法。

2.用传统流程图表示以下问题的算法。

（1）判断两个数中的最大数。

（2）将两个数进行交换。

（3）判断一个年份是否是闰年。

（4）判断一个数是否是素数。

（5）求 1+2+3+……+100 的值。

（6）输入一个华氏温度，要求输出摄氏温度，公式为

$$C = \frac{5}{9}(F - 32)$$

3.写出以下程序的运行结果，并调试运行。

（1）

```
#include   <stdio.h>
int main（）
{
   int c = 67 , d = 68 ;
   printf（"%d %d\n" , c , d）;
   printf（"%d , %d\n" , c , d）;
   printf（"%c , %c\n" , c , d）;
   printf（"c = %d , d = %d\n" , c , d）;
return 0;
}
```

（2）以下程序，输入 12 并回车时，输出结果为　　。

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

```
    char ch1,ch2;
    int n1,n2;
ch1=getchar（）;
ch2=getchar（）;
n1=ch1-'0';
n2=n1*10 +（ch1-'0'）;
printf（"%d\n",n2）;
return 0;
}
```

4.判断以下程序的错误并修改，上机验证。

（1）要求结果输出为“a=%d=2,c=%d=3”

```
#include   <stdio.h>
int main（）
{
    int a=2,b=3;
    printf（"a=%d=%d,b=%d=%d\n" , a ,b）;
    return 0;
}
```

（2）要求输出结果为“9，10”

```
#include   <stdio.h>
int main（）
{
    int m=8;n=10;
    printf（"%d,%d\n" , m++ , - -n）;
    return 0;
}
```

项目四 选择程序设计

程序的基本结构是顺序结构，语句自上而下顺序执行，是无条件的，无需作任何判断。但实际上，要解决一个实际问题，往往需要根据条件是否满足来决定语句的执行，或者从给定的两种或多种操作选择其一。这就是选择结构要解决的问题。

任务 4.1 关系运算符和关系表达式

任务目标

1、理解关系运算的意义和作用

2、掌握 6 种关系运算符的使用方法

任务实施

关系运算也就是“比较运算”，是对两个值进行比较，判断结果是否符合指定的条件，如果满足指定的条件，结果为“真”，不满足指定的条件，结果为“假”。例如，total>90，假设 total 的值为 91，则结果为“真”，total 的值为 60，则结果就为“假”。

4.1.1 关系运算符

C 语言提供了 6 种关系运算符：

>	大于
>=	大于或等于
<	小于
<=	小于或等于
==	等于
!=	不等于

关系运算符的优先级

（1）前四种关系运算符（ > 、>= 、 < 、 <= ）的优先级相同，后两种（ == 、 !=）也相同，前四种的优先级高于后两种。

（2）关系运算符的优先级低于算术运算符。

（3）关系运算符的优先级高于赋值运算符。

如：

a=b>c　　等效于 a=（b>c）

c>a+b　　等效于 a>（a+b）

a==b<c　　等效于 a==（b>c）

4.1.2 关系表达式

用关系运算符将两个表达式连接起来的式子，称为关系表达式。关系表达式的值是一个逻辑值，值为“真”或“假”，这个“真”和“假”一般是用 1 和 0 来表示，1 表示“真”，0 表示“假”。

如：5>3 的值为“真”，表达式的值为 1

3<5 的值为“假”，表达式的值为 0

关系运算符连接的式子可以是常量，变量，算术表达式，逻辑表达式，字符表达式等）。

如：下列的表达式都是合法的关系表达式

a+3>b ， ‘x’>’y’ ， （a=1）<=（b=2）

思考以下表达式的值：

如果 a=1，b=2，c=3，则

（1）d=a>b ，d 的值为　　。

（2）f=a<b<c，f 的值为　　。

（3）g=a+b>c，g 的值为　　。

任务 4.2 逻辑运算符和逻辑表达式

任务目标

1、理解逻辑运算的意义和作用

2、掌握 3 种关系运算符的使用方法

任务实施

用逻辑运算符将关系表达式或逻辑量连接起来的式子就是逻辑表达式。例如年龄限制在 14 到 16 岁之间的逻辑表达式，就可以用以下式子来表示

age >= 14　and　age <= 16

4.2.1 逻辑运算符

C 语言提供了三种逻辑运算符：

&&　　逻辑与（即 AND，表示并且）

||　　逻辑或（即 OR，表示或者）

!　　逻辑非（即 NOT，表示相反）

其中，&&和||是双目运算符，要求有两个操作数，！是单目运算符，只要求有一个操作数。

如：

（x > y）&&（ a> b）、（x>y）||（a>b）、 !a

表 4.1 为逻辑运算的真值表，“1”代表“真”，“0”代表“假”。

表 4.1 逻辑运算真值表

0&&0	0	即有 0 为 0，全 1 为 1
0&&1	0	
1&&0	0	
1&&1	1	
0||0	0	即有 1 为 1，全 0 为 0
0||1	1	
1||0	1	
1||1	1	
!0	1	即 0 取反为 1，1 取反为 0
!1	0	

逻辑运算符的优先级为

（1）！ 高于 && 高于 ||

（2）&& 和 || 低于关系运算符 ！高于算术运算符。

4.2.2 逻辑表达式

用逻辑运算符连接的式子称为逻辑表达式。逻辑表达式的值也是一个逻辑值，即为“真”或“假”，用 1 表示“真”，用 0 表示“假”。

注意：关系表达式或逻辑表达式的值，用 1 表示“值”，但反过来，在判断一个值是否为“真”时，是用非 0 来表示“真”，用 0 就表示“假”。

如：3 > 2 && 5 > 2 的值为 1

!（3>2）的值为 0

!3 的值为 0

思考以下表达式的值：假设 a=1，b=2，c=3

表 达 式	值
a+b>c && b==c	
a || b && b-3	
!a && b&&c	
a+b && b-c &&0	

在逻辑表达式的运算中，并不是所有的逻辑运算符都要被执行，当前面的执行已经可以得到结果时，后面的操作将不会被执行。例如，假设 A、B、C 分别表示一个表达式，则分为以下几种情况：

（1）A || B || C　　当表达式 A 为“真”时，表达式 B 和表达式 C 都不会被执行，因为不管后面两个表达式的值为“真”或“假”，整个表达式的值都为“真”。当表达式 A 为假时，则要执行表达式 B，如果表达式 B 为“真”，则表达式 C 将不会被执行，只有当表达式 A 和表达式 B 都为“假”时，表达式 C 才会执行。

（2）A && B && C　当表达式 A 为“假”时，表达式 B 和表达式 C 都不会被执行，因为不管后面两个表达式的值为“真”或“假”，整个表达式的值都为“假”。当表达式 A 为“真”时，则要执行表达式 B，如果表达式 B 为“假”，则表达式 C 将不会被执行，只有当表达式 A 和表达式 B 都为“真”时，表达式 C 才会执行。

也就是说，当一个表达式已经能确定一个结果时，后面的运算都不会被执行。熟练掌握关系运算符和逻辑运算符后，可以用一个表达式来表示复杂的条件。

任务 4.3　if 语句

任务目标

1、理解 if 语句的意义和作用

2、掌握 if 语句的三种结构

3、掌握用 if 语句编写相应程序

任务实施

if 语句也就是选择语句，用来进行条件判断，根据判断的结果（“真”或“假”），来决定执行的语句。

4.3.1　if 语句的三种形式

C 语言提供了三种形式的 if 语句：

（1）单分支语句：

if （条件表达式）　语句 A；

条件表达式可以是关系表达式，也可以是逻辑表达式。如果条件成立，将执行语句 A，如果条件不成立，将不会执行语句 A，而从 if 语句的下一条语句执行。

但注意，只要条件表达式的值为 0，就表示条件不成立，非 0，就

表示条件成立。

例如：

if（a>b）printf（“变量 a 的值大于变量 b 的值”）;

这种 if 语句的流程图见图 4-1。

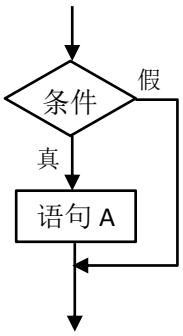


图 4-1 单分支结构

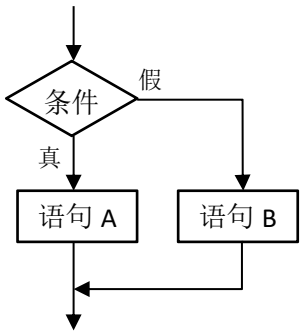


图 4-2 双分支结构

（2）双分支语句：

if （条件表达式）

语句 A;

else

语句 B;

如果条件成立，执行语句 A，条件不成立，执行语句 B。流程图见图 4.2

例如

if（a>b）

printf（”a 比 b 大。”）;

else

printf（”a 比 b 小。”）;

注意：语句 A 和语句 B 可以是一条语句，也可以是用{ }括起来的复合语句。双分支结构的流程图见图 4-2。

（3）分支嵌套语句

if（条件表达式 1）

语句 A；

else if（条件表达式 2）

语句 B；

else if（条件表达式 3）

语句 C；

…　　…

else　语句 n;

这种结构就是在 if 中或 else 中的语句又是一个分支语句。

说明：

1. 三种形式的 if 语句中，在 if 后面都有“条件表达式”，一般为关系表达式或逻辑表达式。系统对表达式的值进行判断，如果为 0，则认为条件为“假”，如果为非 0，则认为条件为“真”。

2. 特殊情况下，条件表达式可以是一个具体的值，。而不是关系表达式或逻辑表达式，此时，直接根据表达式的值是 0 或非 0 来判断为“真”或“假”。如：

if（9）　printf（“It’s OK!”）;

则将会在屏幕上输出“It’s OK!”

另外，表达式的这个值也不限于数值型，也可以是字符型等其他类型。例如，将上面的语句改为 if（’a’） printf（“It’s OK!”）；也会执行相同的结果。

3. if 语句中，条件表达式后面不能用分号，else 后面也不能有分号，其它的语句后面则一定要有分号。

4. 实际上 if 和 else 之间的语句往往不止一条，则要用{ }将这些语句括起来，作为复合语句。注意，复合语句里的每一条语句都要有分号，但是复合语句花括号外不要加分号。

例 4.1 从键盘上输入两个数，将这两个数从大到小输出。

```
  #include   <stdio.h>
main（）
    {
     float a1,a2,t;   //有可能输入小数，t 是中间临时存放数据的。
     scanf(“%f,%f”,&a1,&a2);
     if(a1<a2)
     {t = a1 ; a1 = a2 ; a2 = t ;}
     printf（“% f , % f” , a1 , a2）;
}
```

例 4.2 输入三个数，按从大到小顺序输出。

```
      #include   <stdio.h>
main（）
        {
         float a1 , a2 , a3 , t ;
         scanf（“%f,%f,%f”,&a1,&a2,&a3）;
         if（a1 < a2）
           { t = a1 ; a1 = a2 ; a2 = t ; }
         if（a1 < a3）
           { t = a1 ; a1 = a3 ; a3 = t ; }
         if（a2 < a3）
           { t = a2 ; a2 = a3 ; a3 = t ; }
         printf（“% f , % f , % f ” , a1 , a2 , a3）;
}
```

从上面两个例子可以看出，当数据增加时，代码的复杂程度将会成倍增加。

例 4.3 输入一个年份，判断其是否为闰年。

闰年的条件是：能被 4 整除并且不能被 100 整除；或者能被 400 整除。

```
#include <stdio.h>
main()
 {
 int year;
 if（（year % 4 == 0 && year % 100 !=0）||  （year % 400 == 0））
      printf（“ %d 是闰年。”, year）;
 else
    printf（“%d 不是闰年。”, year）;
}
```

4.3.2 条件运算符

条件运算符是 c 语言中唯一个三目运算符，它要求有三个操作对象，可以用它来代替简单的 if 双分支结构。

条件运算符的一般形式如下：

条件表达式 ？ 语句 A ： 语句 B

例如，以下语句：

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

可以用条件运算符来表示：

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

说明：

（1）条件运算符的优先级高于赋值运算符，所以，上面的式子中，

先执行条件表达式，得到一个值，再将这个值赋值给变量 max，即 max 的值是 a 和 b 中的最大值。

（2）条件运算符的结合方向是“自右向左”。

（3）条件表达式不能完全代替 if 语句，只有在 if 语句中内嵌的语句为赋值语句，并且两个分支都给同一个变量赋值时，才能代替 if 语句。例如下面的 if 语句就不能用一个条件表达式来代替。

```
if（a > b）
   printf（“ % d 大于 % d ”, a , b）;
else
   printf（“ % d 小于 % d ”, a , b）;
```

（4）条件表达式中，条件表达式可以和语句 A 和语句 B 的数据类型可以不一致。如：

```
intx ? ’ c ’ : ’ d ’
```

如果整形变量 intx 的值为非 0，则表达式的值为字符’c’。

实际上，语句 A 和语句 B 的数据类型也可以不一致。如：

```
a > b ? 1 : 2.5
```

如果 a>b 成立，则表达式的值应该为整型值 1，但是因为 2.5 是实型，比整型高，因此，整型 1 会变自动转换成实型 1.0。

任务 4.4 多分支语句 switch

任务目标

1、理解 switch 语句的作用

2、掌握 switch 语句的结构和使用方法

任务实施

if 语句只有两个分支可供选择，如果要进行多分选择，就要用到 if 的嵌套，随着分支数目的增多，if 语句的嵌套就会增加，程序变得

冗长，且可读性降低。为此，C 语言提供了多分支语句 switch，可以使程序精简、便于阅读。

switch 语句的一般形式如下：

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

说明：

（1）switch 后面的括号中的表达式，其值一般为整数或字符型。

（2）每一个 case 后面的常量的值必须不相同，否则，就会出现相互矛盾的现象。

（3）当表达式的值与某一个 case 后面的常量值相同时，就执行此 case 语句后面的语句，如果没有相匹配的常量值，则执行 default 后面的语句。

（4）当执行了一个 case 语句后，会依次向下继续执行后面其它 case 中的语句，想要在执行某个 case 语句后，退出 switch 语句，则要使用 break 语句。

例：根据成绩等级，打印成绩的分数段。

```
swithc（grade）
{
    case   ‘A’：printf（“90~100\n”）;
case   ‘B’：printf（“70~89\n”）;
```

case ‘C’：printf（“60~69\n”）;

case ‘D’：printf（“<60\n”）;

default：printf（“error!\n”）;

}

如果 grade 的值为“B”，则将输出以下值：

70~89

60~69

<60

error!

显然不符合要求，所以，要在执行了某一个 case 后面的语句后退出 switch，则要加上 break 语句。因此，上面的代码修改为：

swithc（grade）

{

case ‘A’：printf（“90~100\n”）; break;

case ‘B’：printf（“70~89\n”）; break;

case ‘C’：printf（“60~69\n”）; break;

case ‘D’：printf（“<60\n”）; break;

default：printf（“error!\n”）;

}

最后的 default 语句由于是 switch 中的最后一条语句，所以不用加 break 语句了。

（5）各个 case 标号出现的顺序并不影响执行的结果，例如可以先出现 case ‘C’，再出现 case ‘A’，甚至 default 可以出现在 case ‘A’ 前面，只是为了便于阅读程序，一般按顺序写每一个 case 语句。

（6）case 只起标记的作用，并不在此进行条件检查。

（7）多个 case 可以共同执行一组相同的语句。

例 4.4 有一个函数如下：

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

从键盘输入一个整数 x，输出 y 的值。

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

项目小结

本任务主要介绍了关系运算和逻辑运算，以及在 C 语言中如何使用 if 语句和 switch 语句来完成选择结构的程序设计，通过本项目的学习，要掌握选择结构的程序设计方法，掌握绘制流程图的方法，能解决有关选择结构的实际问题。

习题

1. C 语言中，如何表示“真”和“假”?系统如何判断一个值是“真”或“假”？

2. 有 4 个不同的整数，请找出最小的数，并输出。

3. 输入一个整数，判断能否同时被 2 和 3 整除。画出对应的流程图，并编写相应程序。

4. 输入一个温度值，如果超过 28，则提示“温度过高，请注意降温!”，否则提示“当前温度正常，请放心使用!”。画出对应的流程图，并编写相应程序。

5. 输入一个圆的半径（r），当 r >= 0 时，计算并输出圆的面积和周长，否则，提示输入错误。

6. 进入 C 的密码。程序运行时，提示输入密码，如果输入正确，显示“欢迎进入 C 程序”，否则，显示“密码输入错误，现在退出”。

7. 输入一个成绩，来输其等级，等级标准为，90 分以上为“优”，70~89 为“良”，60~69 为“及格”，0~60 为“不及格”，其余数值为无效输入值，提示“input error!”。画出流程图，并编写相应程序。

8. 输入一个不多于 5 位的正整数，判断其是几位数?求出各位上的数字，并输出。

9. 多用户密码。有三个用户，输入对应的密码，显示相应提示信息，否则，显示错误信息。分别用 if 语句和 switch 完成。

10. 编程设计一个简单的计算器程序。从键盘输入 2 个操作数，1 个运算符，当运算符为加（+）、减（-）、乘（*）、除（/）时，输出计算结果。

11.个人所得税计算，应纳税款的计算公式如下：

<=5000　　0%

（5000 – 8000]　　3%

（8000 – 17000]　　10%

（17000 – 30000]　　20%

（30000 – 40000]　　25%

（40000 - 60000]　　30%

（60000 – 85000]　35%

> 85000　　45%

输入某人的收入，计算出应纳税额及实际得到的报酬。

项目五 循环程序设计

前面我们学习了顺序结构和选择结构，它们是程序设计中非常重要的结构，但只有这两种结构是不够的，在解决实际问题时，很多地方都要用到循环结构。循环结构是结构化程序的三种基本结构之一，它和顺序结构、选择结构一起，组成程序的基本单元。要解决复杂的问题，就必须熟练掌握这三种结构。

在 C 语言中，可以用以下语句来实现循环：

1. goto 语句和 if 语句一起构成循环；

2. 用 while 语句；

3. 用 do…while 语句；

4. 用 for 语句。

由于 goto 语句和 if 语句构成的循环不便于程序的阅读和修改，建议不用这种结构来编写循环程序，因此，就不予介绍。

任务 5.1　while 语句

任务目标

1、掌握 while 语句的基本语法，理解 while 循环的流程

2、能用 while 语句来编写循环结构程序

任务实施

while 语句用来实现“当型”循环。

其特点是先判断，后循环。

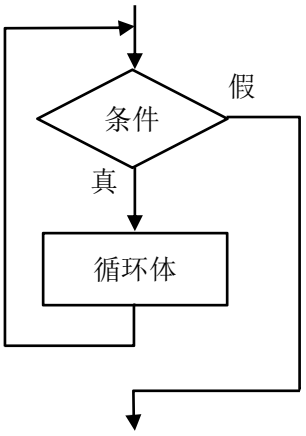


图 5-1 while 循环流程图

一般形式如下：

```
while（条件表达式）
  {
      循环体；
}
```

当条件表达式为真（非 0）时，执行循环体的语句；当条件表达式为假（0）时，退出循环。流程图见图 5-1。

说明：

（1）循环体可以是一条语句，也可以是复合语句。如果循环体只有一条语句，可以不用花括号。

（2）while 后面没有分号。

（3）循环体的次数是由循环条件控制的，所以循环体里应该有使循环趋于结束的语句，避免出现无限循环（死循环）。

例 5.1　求 1 + 2 + 3 + … … + 100 = ?

分析：

这是一个累加的例子，每一个累加的数都比上一个数多 1。我们需要用到两个变量，一个变量 s 用来存放累加的和，一个变量 n 用来表示每一次要累加的数，n 的初始值为 1。用一个循环，每一次累加后，n 的值都加 1，只要 n 的值没有超过 100，就一直累加下去。流程图见图 5-2。

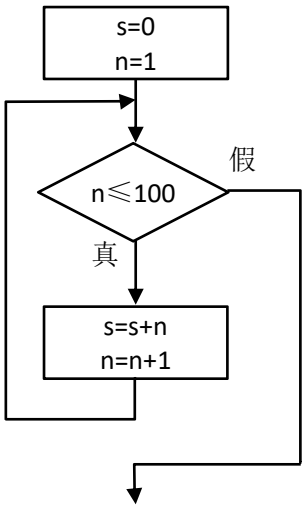


图 5-2 累加程序流程图

程序如下：

```
#include <stdio.h>
main（）
{
 int s = 0 , n = 1;     //定义变量 s 和 I,并赋初值
 while（n <= 100）
  {
   s = s + n ;       //累加
   n = n + 1 ;
//循环变量+1，如果没有这一句，将会是一个无限循环。
}
 printf（”1 + 2 + 3 + … … + 100 = ” , s）;   //输出结果
}
```

思考：

（1）变量 s 和 i 如果没有赋初值，会得到正确的结果吗?

（2）如果去掉循环体中的花括号，会出现什么情况?

例 5.2 从键盘上输入数值，如果不为 0，就进行累加，当输入为 0，程序结束。

分析：

这仍然是一个累加的例子，只不过累加的值是从键盘上输入的，

所以需要用到 scanf 函数，循环的条件是输入的值不为 0，由于 while 循环是先判断后循环，所以，在循环之前要输入值给循环变量 n，累加的和依然存放到变量 s 中。

程序如下：

```
#include <stdio.h>
main（）
{
 float s = 0.0 , n ; //有可能输入小数
 scanf（”请输入数值：% f ”, & n）；  //输入 n 的值
 while（n ! = 0.0）
 {
  s = s + n;
  scanf（”请输入数值：% f”, &n）;
}
  printf（”累加的和为:”, s）；  //输出结果
}
```

任务 5.2　do… while 语句

任务目标

1、掌握 do … while 语句的基本语法

2、能用 do…while 语句来编写循环结构程序

任务实施

do…while 语句用来实现“直到型”循环。其特点是先循环，后判断。

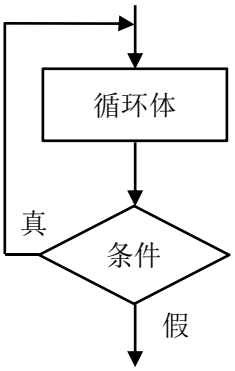


图 5-3 do…while 循环流程图

一般形式如下：

```
do
{
 循环体；
}while（条件表达式）
```

执行过程是先进入循环体，条件表达式为真（非 0）时，返回循环体继续循环，至到条件表达多的值为假（0）时，循环结束。流程图见图 5-3。

例 5.3 用 do…while 语句来实现 1 + 2 + 3 + … + 100 = ?

流程图见图 5-4

程序如下：

```
#include   <stdio.h>
main（）
{
   int s = 0 , n = 1 ;
   do
    {
     s = s + n ;
     n = n + 1 ;
    }while（n <= 100）
    printf（”1 + 2 + 3 + … … + 100 = ” , s）;
}
```

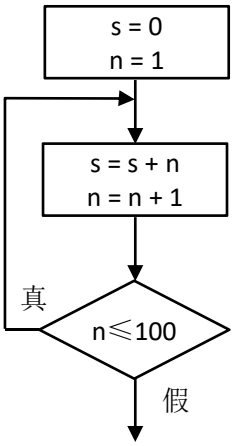


图 5-4 do…while 循环实现累加和

说明：

（1）从例 5.1 和 5.3 可以看出，对同一个问题，可以用 while 循环实现，也可以用 do…while 循环实现，两者完全可以互换。

（2）while 循环中的循环体有可能一次都不执行，而 do…while 循环中的循环体至少执行一次。

任务 5.3　for 语句

任务目标

1、掌握 for 语句的基本语法，理解 for 循环的流程

2、能用 while 语句来编写循环结构程序

任务实施

C 语言中的 for 语句是最灵活的循环语句，当不仅可以用于循环次数确定的情况，而且还可以用于循环次数不确定而只给出循环结束条件的情况。

for 语句的一般形式为：

for（表达式 1；条件表达式 2；表达式 3）

{

循环体;

}

执行过程如下：

（1）执行表达式 1；

（2）判断条件表达式 2 的值，如果为真（非 0），就执行循环体；然后执行表达式 3，转回条件表达式 2，重复执行。

（3）如果条件表达式 2 的值为假（0），则退出循环。

（4）退出循环。

for 循环流程图见图 5-5。

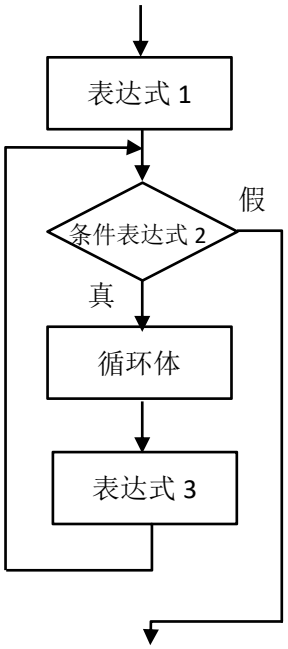


图 5-5　for 语句流程图

for 循环中三个表达式的主要作用为：

表达式 1：可以为零个、一个或多个变量赋初值;

条件表达式 2：循环条件，用来判定是否进行循环。

表达式 3：一般为改变循环变量的值，用来保证循环能够结束。

最常见的 for 语句结构如下：

```
for（循环变量赋初值;循环条件;循环变量增减值）
{
 循环体；
}
```

例如：求 1 到 100 的累加和，也可以用以下代码实现：

```
for（n = 1 ; n <= 100 ; n++）
{
 s = s + n;
}
```

显然，用 for 语句更简单，更方便，更便于阅读。

说明：

（1）for 语句中的语句 1 可以省略，但是不能省略分号。

如：for（ ; n <= 100 ; n++） s = s + n ;

此时，表达式 1 被省略，循环执行时，将忽略执行表达式 1 这一步骤。

（2）条件表达式 2 也可以省略，即循环不判断条件，将无限循环下去。

（3）表达式 3 也可以省略，但应该另外设计代码，保证循环能正常结束。

例 5.4 输出 100 以内能被 3 整除的所有数。

```
#include   <stdio.h>
main（）
{
 for（n = 3 ; n < 100 ; n++）
   {
    if（n % 3 == 0）
       printf（” % d \n” , n）;  //每行打印一个能被 3 整除的数。
}
}
```

任务 5.4 循环的嵌套

任务目标

1、掌握嵌套循环的基本含义

2、能编写多重循环结构的程序

任务实施

一个循环体中的语句又包括一个循环，称为循环的嵌套，循环里又有循环。while 循环、do…while 循环、for 循环可以互相嵌套，而且可以多次嵌套。

例如：以下几种嵌套都是正确的。

（1）for（ ； ； ）
{
for（; ;）
{ … }
}

（2）for（ ； ； ）
{
while（）
{ … }
}

（3）while（）
{
for（ ； ； ）
{ … }
}

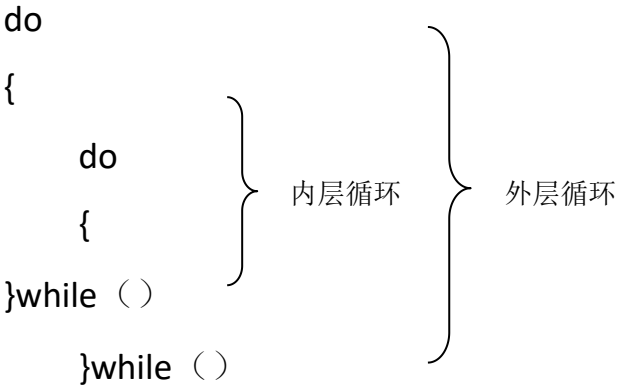
（4）while（）
{
while（）
{ … }
}

（5）do
{
while（）
{ … }
}while（）

（6）do
{
for（ ; ; ）
{ … }
}while（）

当然，循环的嵌套不止这 6 种，这三种循环语句可以任意嵌套。使用循环嵌套时，要注意循环的嵌套关系，即内层的循环必须完整包括在外层循环里，不能交叉。

类似下面这种嵌套就是错误的，一定要避免。



任务 5.5　break 语句和 continue 语句

任务目标

1、理解 break 和 continue 语句的作用

2、能用 break 和 continue 语句来编写相应循环结构程序

任务实施

正常情况下，循环体里的语句根据依次执行，至到循环条件不满足才退出循环。但是，在某些特殊情况下，需要提前终止循环。例如在进行 1 到 100 的累加时，如果累加到 1500 就要结束时，循环条件是〈=100，但当累加的值超过 1500，就不再进行循环，而就当结束循环了。这种改变循环执行状态的操作可以用 break 语句和 continue 语句来实现。

5.5.1　break 语句

之前在介绍 switch 语句时，已经知道 break 语句，可以中止 switch 语句的执行，跳转到 switch 语句之后的语句继续执行程序，实际上，break 语句也可以终止循环执行，它能跳出循环，转到循环语句之后的语句继续执行。

例 5.5 全校共有 2000 名师生进行募捐，当募捐总数达到 5 万元时就停止，统计此时捐款的人数及平均捐款数。

分析：师生总数为 2000 名，所以最多可以进行 2000 次捐款，但不确定第几名师生的捐款总数会达到 5 万，所以，在循环体中，加上一个判断，超过 5 万，就停止循环。变量 s1 来保存师生每一次捐款数，变量 s2 来保存捐款总数，变量 aver 来保存平均捐款数，变量 n 来控制循环次数。

程序如下：

```
#include   <stdio.h>
main（）
{
   float s1 = 0 , s2 = 0 , aver = 0 ;
   int n ;
for（n = 1 ; n <= 2000 ; n ++）
 {
printf（”请输入捐款数:”）; //屏幕提示信息
scanf（” % f ” , & s1）;   //输入捐款数
s1 = s1 + s2 ;    //累加
if（s1 >= 50000） break ;
//如果累计超过 50000，就退出循环
          }
        aver = s1 / n ;    //求平均捐款数
        printf（”捐款人数是：% d \n 平均捐款数是：% f” , aver , n）;
}
```

说明：

（1）break 语句只能出现在 switch 语句和循环语句中，不能出现

在其它地方。

（2）break 语句在循环语句中，一般会加上条件，所以，经常会和 if 语句一起使用。

（3）break 语句是跳出循环，接着从循环体的下一条语句继续执行。

（4）注意：如果 break 语句是出现在嵌套循环中，它只能跳出自己本身所在那一个循环，并不会跳出外循环。

5.5.2 continue 语句

有时候并不希望终止整个循环，而是跳过本次循环，进入下一次循环，这就要用到 continue 语句。

continue 语句执行时，循环体中后面的语句将不会被执行，进入下一次循环。

例 5.6 从键盘输入十个整数，判断正数有多少个?

分析：定义一个变量 num，用来保存正数的个数，当输入的是正数时，num 加 1，负数时，num 不加 1。

程序如下：

```
#include   <stdio.h>
main（）
{
   int n , num = 0 , zx ;   //n 是循环变量，zx 用来接收输入的值
printf（”请输入 10 个整数：\n”）;
for（n = 1 ; n <= 10 ; n++）
    {
       printf（”\n 请输入第%d 个整数: ” , n）;
       scanf（”%d” , zx）;
```

```
            if（zx < 0） continue;     //输入的值为负数，跳过 num++，即不计入正整数个数
            num ++ ;
        }
        printf（”输入的 10 个数中，正整数的个数为%d :” , num）;
    }
```

当然，这个例子中的 if 和 num++这两条语句，也可以改成以下的一句代码：

```
    if（zx > 0） num ++ ;
```

这样程序更好阅读，代码更短，效率更高，我们在程序中使用 continue，只不过是为了说明 continue 语句的作用而已。

任务 5.6 程序实例

任务目标

通过实例，巩固编程思路，提高编写程序的能力

任务实施

例 5.7 输出斐波那契数列的前 20 个数。

分析：斐波那契数列是意大利数学家 Fibonacci 提出来的，第一个数为 0，第二个数为 1，从第三个数开始，值等于它前两个数之和。即：斐波那契数列为 0,1,1,2,3,5,8,13,21,34…，算法为：

F1 = 0　　（n = 1）

F2 = 1　　（n = 2）

$Fn = F_{n-1} + F_{n-2}$ （n >= 3）

程序如下：

```
#include   <stdio.h>
main（）
{
   int    f1 , f2 ;
   int    n ;
   f1 = 0 , f2 = 1 ;
   for（n = 1 ; n <= 10 ; n ++）
       {
printf（”% d , \ t %d” , f1 , f2）;
        if（n % 2 == 0） printf（” \ n ”）;  //每输出 4 个值，就换行
        f1 = f1 + f2 ;    //每次循环计算后两个数，这是第一个数
        f2 = f2 + f1 ;    //这是第二个数
       }
}
```

运行结果如下：

```
0       1       1       2
3       5       8       13
21      34      55      89
144     233     377     610
987     1597    2584    4181
```

此例用数组来实现更方便理解和阅读，有关数组的相关知识，将在第六章进行介绍。

例 5.8 判断一个数是否是素数。

分析：所谓素数，是指除了 1 和本身之外，不能被其它数整除。

可以用一个循环，从 2 开始到这个数减 1，依次去除这个数，只要被其中任意一个数整除，就不是素数，否则，就是素数，程序中，可以用一个标志变量 flag，根据其值来判断是否被整除过。

流程图见图 5-6：

程序如下：

```
#include   <stdio.h>
main（）
{
  int n , m , flag = 0 ;
  scanf（”%d” , &m）;
  for（n = 2 ; n < m ; n ++）
     if（m % n == 0）
{
//被整除过，flag 赋值为 1
   flag = 1 ;
break ;
         }
if（flag == 1）
        printf（” % d 不是素数!” , m）;
else
              printf（” % d 是素数! ” , m）;
}
```

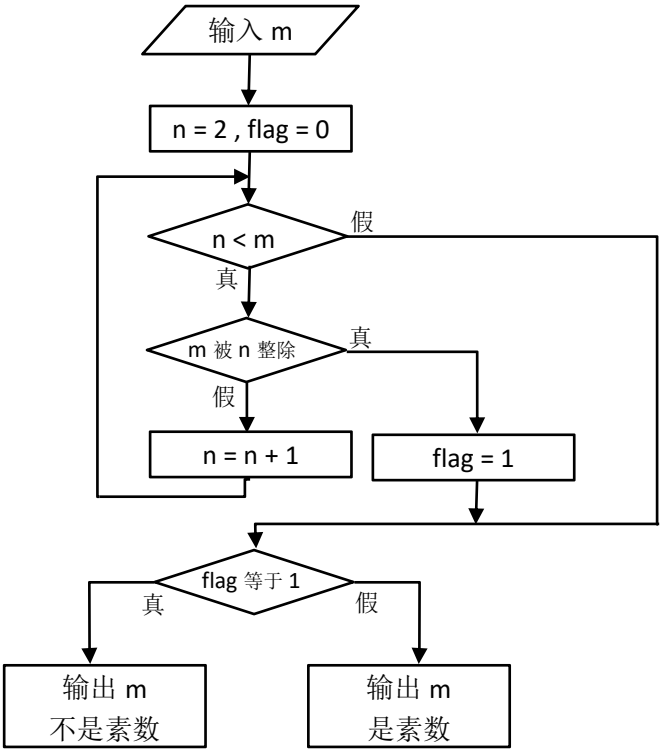


图 5-6 判断素数的流程图

运行结果如下：

```
23
23 是素数!
```

实际上，上面的代码可以优化，判断一个数是否是素数，并不一定要从 2 一直除到本身减 1，可以除到本身除以 2 这个数就可以了，这样，可以使代码在执行的时候，少循环一半的次数，甚至只需要除到这个数的开平方就可以了。大家可以试着去优化一下代码。

例 5.9 显示所有的“水仙花数”。

分析：“水仙花数”是指一个三位数，其各位数字的立方和等于这个数本身。例如 $153=1^3 + 5^3 + 3^3$，所以，153 就是“水仙花数”。思路是：循环从 100 到 999，每循环一次，将各位上的数字分别保存到三个不同的变量，然后计算其立方和，如果等于这个数本身，就是“水

仙花数”，否则不是“水仙花数”。

程序如下：

```
#include   <stdio.h>
main（）
{
   int g , s , b ;
   int n ;
   for（n = 100 ; n <= 999 ; n++）
     {
        b = n / 100 ;   //得到百位数上的数字
        s =  （n – b*100）/ 10 ；      //得到十位数上的数字
        g = n – b * 100 – s * 10 ;   //得到个位数上的数字
        if（n == b * b * b + s * s * s + g * g * g）;
            printf（”% d \ t ”, n）;
}
}
```

请大家自行画出流程图。

项目小结

本项目是编写程序最重要的内容，通过本项目的学习，应掌握循环程序设计的基本方法，能用编程思想解决问题。在解决实际问题的过程中，应根据实际情况，考虑用选择结构还是循环结构，很多时候，选择结构程序和循环结构程序不是孤立的，应该配合起来使用，在编写程序前，应当根据解决问题的思路，绘制出流程图，再根据流程图，编写相应程序。

习题

6.1 凯撒密码。从键盘输入一串英文字母（假设都为小写字母），将每个小写字母都向后移动 2 位，即，’a’转换为’c’，’b’转换为’d’，…‘y’转换为’a’，’z’转换为’b’。

6.2 输入一行字符，分别统计其中的字母、数学、空格以及其他字符的个数。

6.3 输入两个整数 m 和 n，求最大公约数和最小公倍数。

6.4 有 0、1、2、3 四个数字，能组成哪些各位上数字不相同的三位数？一共有多少个组合?

6.5 打印如下图形。

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

6.6 编程输入一个整数，判断是否是素数.

6.7 打印九九乘法表。

6.8 中国古代的“百鸡买百钱”问题。一只公鸡值 5 元钱，一只母鸡值 3 元钱，三只小鸡 1 元钱，问 100 元钱买 100 只鸡，公鸡、母鸡和小鸡各多少只。

6.9 韩信点一队士兵的人数，三人一组余两人，五人一组余三人，七人一组余四人。问：这队士兵至少有多少人？

6.10 一个整数，加上 100 后是一个完全平方数，再加上 168 又是另一个完全平方数，请问该数是多少？（一个正数的平方叫做完全平方数）

项目六 数 组

前面的程序中使用的变量都是基本数据类型，例如整型、浮点型、字符型等。对于简单的问题，用基本的数据类型就能够解决问题，但是对于复杂的问题，只用简单的数据类型就比较困难，例如，统计50名学生的成绩，如果定义50个变量来存放成绩，会使程序显得冗长且效率不高。因此，本章开始介绍一种新的数据类型：数组。

所谓数组，是指一组有序数据的集合，它们中的所有元素都属于同一个数据类型。例如50名学生的成绩，第一名学生的成绩都是float类型。

任务 6.1 一维数组的定义和引用

任务目标

1、理解数组的含义，理解数组在存储器中的保存方式

2、掌握数组的定义方法

3、掌握用一维数组编写程序的基本方法

任务实施

6.1.1 一维数组的定义

一维数组的定义如下：

类型类型　数组名 [常量];

例如 50 名学生的成绩可以定义成如下数组：

float　cj [50];

说明：

（1）常量只有一个，称为一维数组，如果有两个常量（两个方括号），称为二维数组。

（2）数组名的命名规则和变量的名规则相同。

（3）数组名后面是方括号，不能用圆括号。

（4）定义数组时，方括号里只能是常量或符号常量，用来表示数组元素的个数。

以下分别定义了几种不同类型的数组，

int　num [10];　//定义一个整形数组，共 10 个元素

float　cj [15];　//定义一个 float 数组，共 15 个元素

char　zf [26];　//定义一个字符数组，共 26 个元素

6.1.2 一维数组元素的引用

数组要先定义后使用，C 语言规定数组中的元素只能一个一个的

使用，不能同时使用整个数组。

数组元素的引用如下：

数组名[下标]

例如：int　num[10] ;

num [0] = 5 ; //　引用 num 数组中的第一个元素

num [9] = num [2] + num [6] ;

说明；

（1）定义了数组后，数组元素的下标是为 0 ~ n - 1。

上面的 num 数组，这样引用数组元素是错误的，

num [10] = 5 ;　//错误，数组下标超过数组范围

特别注意：定义数组时，下标是表明数组元素的个数，引用数组时，下标是从 0 开始的。

（2）引用数组时，下标可以是整型常量、整型变量或整型表达式，只要这个整型值在数组下标范围内。

如：　num [6] = num [1 + 2] / num [3 + 4] ;

例 6.1　给数组赋值为 0 ~ 9，然后倒序输出。

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

程序结果如下：

```
9  8  7  6  5  4  3  2  1  0
```

程序分析：

第一个 for 循环，是对 num[10]数组中的各元素进行赋值，这个循环结束后，各元素的值如图 6 - 1 所示：

num[0]	num[1]	num[2]	num[3]	num[4]	num[5]	num[6]	num[7]	num[8]	num[9]
0	1	2	3	4	5	6	7	8	9

图 6-1 num[10]数组各元素的值

第二个循环依次从后向前输出各元素的值。

再一次提醒:数组元素的下标是从 0 开始的，定义了 10 个元素的数组，则引用时最大下标为 9。

6.1.3 一维数组的初始化

为了程序的简洁，一般在定义数组的同时，给数组各元素赋值，这称为数组的初始化。

（1）定义数组时对所有数组元素赋值。

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

将数组中各元素的初始值按顺序写在一对花括号里，各值用逗号隔开。花括号里的数据就称为“初始化列表”。

（2）如果初始化时，对所有元素都赋初值，在定义时，可以不指明数组元素的个数。如：

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

则表明数组 a 有 4 个元素。

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

（3）初始化列表的值少于数组元素个数，则只初始化前面的元素。如：

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

对数组的前 4 个元素赋初值，后面 6 个元素的值为 0。

特别说明：在对不同数据类型的数组进行初始化时，如果初始化值个数少于数组元素个数时，未初始化的数组元素会赋给不同类型的值，如整型数值就会赋值 0，字符数组就会赋值'\0'，指针数组就会赋值“NULL”。

6.1.4 一维数组程序举例

例 6.2 用数组计算斐波那契数列的前 20 个数。

在第五章的程序中，已经计算了斐波那契数列，但是不容易理解，现在用数组来计算。

```
#include   <stdio.h>
main（）
{
   int    n ;       //循环变量
   int f [ 20 ] = { 0 , 1 };
   for（n = 2 ; n < 20 ; n ++）
         f [ n] = f [ n-1 ] + f [ n – 2 ] ;
// 从第一个元素开始，值等于前一个元素的值加上前二个元素的值。
for （n = 0 ; n < 20 ; n ++） //输出 20 个元素的值
   {
if（n % 4 == 0） printf（”\n”）;
//每输出 4 个值，就换行
printf（”% d” , f [ n ]）;
    }
}
```

运行结果如下：

```
0       1       1       2
3       5       8       13
21      34      55      89
144     233     377     610
987     1597    2584    4181
```

例 6.3 从 20 个整数中找出最大值和最小值。

分析：由于数据较多，采用数组比较方便，另外定义两个变量 max 和 min，分别存放最大值和最小值。依次从数组中取出每一个元素，分别和 max 和 min 进行比较，如果比 max 大，就放入 max 中，否则 max 的值就不变；如果比 min 的值还小，就放入 min 中，否则 min 的值就不变。20 个值比较完成，max 中就是最大值，min 中就是最小值。

程序如下：

```
#include   <stdio.h>
main（）
{
   int num [ 20 ] ;   // 定义一个数组，存入 20 个整数
   int    n , max = 0 , min = 0 ;     //定义循环变量和最大值、最小值
   for（  n = 0 ; n <20; n++）
{
      printf（” \n 请输入第%d 个整数：” ）;
      scanf（”%d” , & num [ n ]）;   //输入 20 个整数
        //最大值放入 max 中
if（max < num [ n ]）  max = num [ n ] ;
//最小值放入 min 中
         if（min > num [ n ]）  min = num [ n ] ;
```

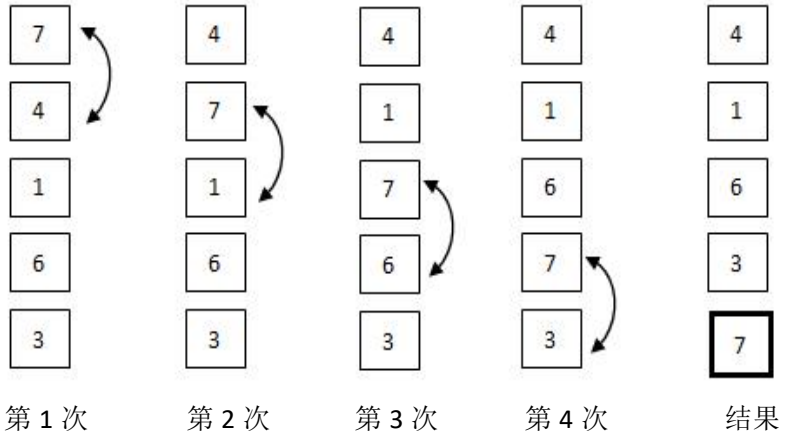
```
    }
printf（”\n 输入的最大值是:%d, 最小值是%d : ”, max , min ）;
}
```

例 6.4 冒泡法将数据由小到大排序。

排序是一种非常重要的一种算法，有很多种排序方法。冒泡法排序是其中的一种方法，基本思路是依次让相邻的两个数进行比较，将较小的数放到前面，一轮交换下来，最大的数已经在最后了，然后重复对前面的数再进行一轮比较，又将最大的数放到最后，依次进行，就可以实现数据由小到大排序。

例如有 5 个数：7，4，1，6，3，在第一轮循环中，第一次将 7 和 4 交换，第二次将 7 和 1 交换，依次进行 4 次，得到新的顺序 4，1，6，3，7。可以发现，最大的值 7 已经跑到最后去了。第二轮循环，对剩下的数 4，1，6，3 进行同样的比较和交换，得到 4，1，3，6。这样 6 也已经排好序了，再进行第三轮循环、第四轮循环，就可以实现排序功能了。

冒泡法排序的示意图见图 6-2。



根据分析可知，有 5 个数，则要进行 4 次循环，才能将这 5 个数按顺序排列，在第 1 次循环中，要进行 4 次比较，第 2 次循环进行 3 次比较… …第 4 次循环只需要进行一次比较。

程序如下：

```
#include   <stdio.h>
main（）
int p[ 10 ] = {4,3,6,8,9,12,32,51,2,4} ;   //假设有 10 个数排序
int n , m , t ;
for（n = 0 ; n < 9 ; n ++）//n 确定需要进行几轮比较
    for（m = 0 ; m < 9 – n ; m ++） //m 确定每一轮比较几次
       if（p [ m ] > p [ m + 1 ]）
          { t = p [ m ] ; p [ m ] = p [ m + 1] ; p [ m + 1 ] = t ; }
printf（”从小到大排序结果如下：\n ”）;
for（n = 0 ; n < 10 ; n ++）
      printf（” %d     ” , p [ n ]）;
}
```

任务 6.2 二维数组的定义和引用

任务目标

1、理解二维数组的含义，掌握二维数组的定义方法

2、掌握用二维数组编写程序的基本方法

任务实施

6.2.1 二维数组的定义

二维数组定义的一般形式为：

数据类型　数组名 [常量][常量]

例如： int　x [2][3] , y [10] [5] ;

定义 x 为 2×3（2 行 3 列）的数组，定义 y 为 10×5（10 行 5 列）

的数组。

C 语言对二维数组采用这样的定义方式，使得二维数组可以看作一种特殊的一维数组，它的元素又是一个一维数组。例如，可以把 x 看成一个一维数组，它有 2 个元素：x[0] ，x [1] 。这两个元素又分别包含 3 个元素的一维数组，如下：

x [0] - - x [0] [0] x [0] [1] x [0] [2]

x [1] - - x [1] [0] x [1] [1] x [1] [2]

即把 x [0] , x [1]当作一维数组的名字。

C 语言中，二维数组中元素排列的顺序是按行连续存放的，即在内存中，先存放第一行的元素，再存放第二行的元素。

第一行			第二行		
x [0] [0]	x [0] [1]	x [0] [2]	x [1] [0]	x [1] [1]	x [1] [2]

二维数组逻辑上是一个矩阵，只是用来表示行列关系，实际在内存中，是连续存放的。

6.2.2 二维数组元素的引用

二维数组元素的表示引用方法是：

数组名 [下标] [下标]

下标可以是整型常量，也可以是整型表达式，如：x [1][3-2] ，注意不要写成 x[1, 3-2]

特别注意：二维数组元素的下标也都是从 0 开始的，在引用二维数组元素时，每个下标的值都不能超过定义数组时的常量-1。

如：

```
int a [ 4 ] [ 5 ] ;      //定义了 4 × 5 的二维数组
a [ 0 ] [ 0 ] = 1 ;     //给第一个元素赋值为 1
```

```
a [ 3 ] [ 4 ] = 20 ;   //给最后一个元素赋值为 20
a [ 4 ] [ 5 ] = 20 ;   //错误，没有 a [ 4 ] [ 5 ] 这个元素
```

6.2.3 二维数组的初始化

用“初始化列表”进行初始化。

（1）分行给二维数组赋初值。例如：

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

这种方法能直观的看到二维数组中各元素的赋值情况。

（2）可以将所有数据写在一个花括号里面，将会按二维数组在内存中的排列顺序对各元素赋初值。

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

与前面的效果是一样的。但是这种赋值方法容易遗漏且不易检查和阅读。

（3）可以只对一部分元素赋初值。

```
int a [ 2 ] [ 3 ] = { {1 ,} , { 0 , 5 , 6 } } ;
```

这样赋值后，数组各元素的值为：

```
1   0   0
0   5   6
```

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

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

这样也是可以的，系统编译时，会知道数组有 3 列，也就知道应该是 2 行。

6.2.4 二维数组程序举例

例 6.5 矩阵转置。将一个二维数组的行、列互换，保存到另一个数组中。

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

分析：定义 a 数组为 2 行 3 列，再定义 b 数组为 3 行 2 列（无需赋值），只要将 a 数组中的元素 a[m] [n]赋值给 b 数组的元素 b [n] [m]就可以了。由于 a 数组是二维数组，用二层循环即可实现。

程序如下：

```
#include   <stdio.h>
main（）
{
   int a [ 2 ] [ 3 ] = { { 1 , 2 , 3 } , { 4, 5 , 6 } } ;
   int b [ 3 ] [ 2 ] ;
   int m , n ;   //循环变量
   printf（”数组 A 为：\n”）;
   for（m=0 ; m < 2 ; m ++）
      {
 for（n= 0 ; n < 3 ; n ++）
       {
         printf（”%10d” , a [ m ] [ n ]）;
 //输出 a 数组的各元素
         b [ n ] [ m ] = a [ n ] [ m ] ;
//将 a 数组中的元素保存到 b 数组中
}
printf（” \n”）;    //输出一行后换行显示
       }
     printf（”数组 B 为：\n”）;      //输出 B 数组
for（m = 0 ; m < 3 ; m ++）
```

```
    {
        for（n = 0 ; n < 2 ; n ++）
            printf（”%10d” , b [ m ] [ n ]）;
printf（” \n”）;    //输出一行后换行显示
    }
}
```

运行结果如下：

```
数组 A 为：
1       2       3
4       5       6
数组 B 为：
1       4
2       5
3       6
```

例 6.6 从一个 4×5 的矩阵中，求出最大的值，以及其所在行号和列号。

分析：分别定义 3 个变量 max，row，col，用来存放最大值，以及行号和列号。开始假设数组中第一个元素是最大值，将其值放入 max 中，行号放入 row 中，列号放入 col 中，再依次用 max 去和数组元素的其他值进行比较，只比 max 值大，就放入 max 中，并记下行号和列号，比较结束，max 中就是最大值，row 中就是行号，col 中就是列号。这种算法称为“打擂台算法”。

程序始下：

```
#include   <stdio.h>
main（）
{
    int row=0,col=0 , max ;
    int m , n ;
```

```
    int num[ 4 ][ 5 ]= { {5,12,26,74,6},{11,-2,41,6,23 },{0,-6,
62,45,20},{16,22,-7,9,50} };
    max= num [ 0 ] [ 0 ] ;
    for（m = 0 ; m < 4 ; m ++）
        for（n = 0 ; n < 5 ; n ++）
            if（num [ m ] [ n ] > max）
              { max = num [ m ] [ n ] ; row = m ; col = n ; }
    printf（”最大值是：%d\n 所在行是：%d \n 所在列是：%d “ ,
max , row , col）;
}
```

任务 6.3 字符数组

任务目标

1、掌握字符数组的定义方法

2、理解字符数组和其他数组的区别

3、掌握用字符数组编写程序的基本方法

4、掌握常用数组函数的使用方法

任务实施

用来存放字符数据的数给称为字符数组。字符数组中的一个元素存放一个字符。另外，C 语言中没有字符串类型，也没有字符串变量，字符串存放在字符组中。

6.3.1 字符数组的定义

字符数组的定义方式和数值型数组的方法类似，格式如下：

```
char c [ 5 ];     //定义包含 5 个元素的字符数组
c [ 0 ] = ‘ a ’ ;     //给数组的第一个元素赋值为字符 ’a’ 。
```

由于字符型数据是以 ASCII 码存放的，也可以用整形数组来存放字符数据，但不提倡。

如：

```
int   a [ 10 ] ;
a [ 0 ] = ‘ a ‘ ;  //正确，但不提倡这样使用
```

6.3.2 字符数组的初始化

对字符数组，采用“初始化列表”的方式是最直观的。如：

```
char c [ 11 ] = { ‘H’ , ’ e ‘ , ‘ l ‘ , ‘ l ‘ , ‘ o ‘ , ‘  ‘ , ‘ w ‘ , ‘ o ‘ , ‘ r ‘ , ‘ l ‘ , ‘ d ‘ } ;
```

将 10 个字符依次赋值给字符数组的 c [0] ~ c [9] 这 10 元素个。

初始化字符数组时，如果初值个数大于数组长度，会出现语法错误；如果初值小于数组长度，则将字符依次赋值给字符数组中的元素，其余的元素自动赋值班为空字符，即’\0’。

如果在定义字符数组时，忽略数组长度，则根据初值个数来确定数组的长度，如：

```
char a [ ] = { ‘ t ’ , ‘ h ‘ , ‘ a ‘ , ‘ n ‘ , ‘ k ‘ } ;
```

则数组的长度自动设置为 5

与前面所讲的二维数组一样，也可能定义和初始化二维字符数组，如：

```
char   cc [5 ] [ 5 ] = { { ‘  ‘ , ‘‘ , ‘ * ‘ } ,
                         { ‘  ‘ , ‘ * ‘ , ‘‘ , ‘ * ‘ } ,
                         { ‘ * ‘ } , ‘‘ , ‘‘ , ‘‘ , ‘ * ‘ } ;
                         { ‘  ‘ , ‘ * ‘ , ‘‘ , ‘ * ‘ } ,
                          { ‘  ‘ , ‘‘ , ‘ * ‘ } } ;
```

字代表一个钻石形的平面图形，如图 6-3 所示。

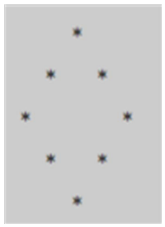
图6－3 钻石平面图

6.3.3 字符数组的引用

字符数组的元素引用方法和整型数组元素的引用方法相同，即用数组名和下标的方式。字符数组下标也是从0开始的。

例6.7 输出钻石图形。

```
#include   <stdio.h>
main（）
{
   char   cc [5 ] [ 5 ] = { {‘  ‘,‘‘,‘*‘},
                      {‘  ‘,‘*‘,‘‘,‘*‘},
                      {‘*‘},‘‘,‘‘,‘‘,‘*‘};
                      {‘  ‘,‘*‘,‘‘,‘*‘},
                       {‘  ‘,‘‘,‘*‘}};
    int m , n ;
    for（m = 0 ; m < 5 ; m ++）
        {for（n = 0 ; n < 5 ; n ++）
            printf（” %c “ , cc [ m ] [ n ]）;
        printf（” \n ”）;    //一行打印结束换行
        }
}
```



运行结果如下：

6.3.4 字符串和字符串结束标志

前面已经学过，字符和字符串是不同的表现形式，在 C 语言中，将字符串作为字符数组来处理。’ a ‘表示一个字符，“a”表示一个字符串，（字符是用单引号，字符串是用双引号）字符串是按顺序依次保存到字符数组中去的。实际上，字符数组的长度并不刚好等于字符串的长度，有用的是字符串的实际有效字符，以 C 语言中，规定了“字符串结束标志”，以字符 ’\0’ 表示，即字符串中最开始出现的字符’\0’之前的数据称为字符串的长度，但字符数组中，还必须至少有一个元素来存放字符结束标志’\0’。程序中，往往以’\0’来表示字符串结束，而不是以字符数组的长度来判断，当然，在定义字符数组时，应该保证字符数组的长度始终大于字符串的长度。

之前出现过的输出语句

printf（”hello world!”）;

实际上，在向内存存储时，会自动在字符的后面加上一个’ \0 ‘ ，作为结束标志。在执行此语句时，系统会每输出一个字符，检查一次结束标志，遇到’\0 ‘就停止输出。

明白了字符串的处理方法后，可以用另外的方法来初始化字符数了。例如：

char c [ ] = { “Hello world” } ; // 注意字符数组的长度为 12

甚至可以省略花括号，直接写成

char c [ ] = “Hello world” ;

对比之前的一个字符一个字符的初始化，显示用字符串的方法更符合人们的习惯。

如果初始化字符数组时，数组长度比字符数多，则后面的元素全部’\0’，如下：

char c [10] = “ thank“ ;

则字符数组 c 的前 5 个元素分别为’t’，’h’，’a’，’n’，’k’，第六个元素为’\0’，后面剩下的四个元素也全部为’\0’。如下图所示。

t	h	a	n	k	\0	\0	\0	\0	\0

图 6 - 4 字符串在内存的表示

说明：字符数组并不要求最后一个元素必须为’\0’，而是指系统在处理字符串时，会自动在字符串的最后加上’\0’。为了表示一致，一般在程序中，往住人为的在字符数组中加入’\0’。

例如：

char c [6] = { ‘ t ’ , ‘ h ’ , ‘ a ‘ , ‘ n ‘ , ‘ k ‘ , ‘ \0 ‘ } ;

这样做的好外是：当把这个数组放到一个较长的数组里时，没有’\0’，则会替代较长数组的前 5 个元素，而后面的元素还在，加了’\0’时，会替代前 6 个字符，且输出时，只会输出前 5 个元素，因为遇到’\0’就会结束。

6.3.5 字符数组的输入输出

字符数组的输入可以有两种方法：

（1）逐个字符依次输入或输出，用%c 输入或输出。

```
for（n = 0 ; n < 5 ; n ++）
    scanf（” %c ” , & c [ n ]）;     //c 是包含 5 个元素的字符数组
```

（2）将整个字符串一次输入或输出，用%s 来输入或输出。

输入字符串用以下方法：

scanf（”%s”, c）;

//c 是一个已经定义好了的字符数组，不能用数组元素

注意：由于数组名就已经代表数组的起始地址，所以数组名前面不能加地址符“&”了。

如果用 scanf（）函数实现多个字符串的输入，则字符串之间用空格进行分隔。例如

```
char s1 [ 20 ] ;
scanf（”%s”, s1）;  //只能数组名 s1
```

运行时，输入数据

```
are  you  ready?
```

则 s1 数组中的数据只有前 3 个字符“are”以及 17 个’\0’。因为遇到第一个空格，就认为第一个字符串输入完毕。

为了输入这句英文，可以用三个字符数组来存放。如下：

```
char c1[10 ] , c2 [ 10 ] , c3 [ 10 ] ;
scanf（”%s%s%s”, c1 , c2 ,c3）;
```

运行时，输入数据

```
are  you  ready?
```

则各个数组的元素如下：

c1:	a	r	e	\0	\0	\0	\0	\0	\0	\0
c2:	y	o	u	\0	\0	\0	\0	\0	\0	\0
c3:	r	e	a	d	y	?	\0	\0	\0	\0

输出字符串的方法:

printf（”%s“, c）;　　//c 是字符数组名

实际上的执行过程是：先按数组名找到数组的第一个元素，依次输出，至到遇到'\0'时停止。这比逐个字符输出要方便。

6.3.6 字符串处理函数

C 语言的函数库中提供了字符串处理的函数。使用函数库中的函数，需要引用头文件“string.h”。下面介绍几种常用的字符串处理函数。

1. 输入符串函数 gets（）

一般形式如下：

gets（字符数组）

输入一个字符串到字符数组中，并且该函数有一个返回值，即该字符数组的起始地址。

思考：如何输入的字符个数超过了字符数组定义的大小时，结果会怎样?

2. 输出字符串函数 puts（）

一般形式如下：

puts（字符数组）

将一个字符串输出。字符串以'\0'为结束标志。

注意：用 gets（）和 puts（）函数时，一次只能输入或输出一个字符串。

3. 字符串连接函数 strcat（）

一般形式如下：

strcat（字符数组 1，字符数组 2）

将字符数组 2 连接到字符数组 1 的后面，字符数组 1 的原来的结

束标志'\0'会被替换，只保留字符数组 2 中'\0'，函数的返回值为字符数组 1 的起始地址。

例如：

char s1 [15] = {“ thank ” } ;

char s2 [ ] = {“ you “ } ;

strcat（str1 , str2）;

则输出结果为：thankyou

连接前 s1 和 s2 的值存储如下：

S1	t	h	a	n	K	\0	\0	\0	\0	\0	\0	\0	\0	\0	\0
S2	y	o	u	\0										

连接后 s1 和 s2 的存储情况如下：

S1	t	h	a	n	K	y	o	u	\0	\0	\0	\0	\0	\0	\0
S2	y	o	u	\0										

说明：

（1）字符数组 1 必须定义得足够大，以便能存放下字符数组 2，否则会出现错误。

（2）连接时，第一个字符数组中的'\0'将会被替换。

（3）字符数组 2 也可以是字符串常量，例如：

char　s1[15] = “thank” ;

strcat（s1 , “ you ”）;

和上面的结果是一样的。

（4）可以用 strncat（）函数将字符数组 2 的前 n 个字符连接到字符数组 1 中，但是 n 不能大于字符数组 2 的个数。例如：

strncat（s1,s2,2）；　//将字符数组 s2 的前 2 个字符连接到 s1。

4. 字符串复制函数 strcpy（）

一般形式如下：

strcpy（字符数组 1，字符数组 2）

将字符数组 2 复制到字符数组 1 中。例如：

```
    char s1 [ 10 ] , s2 [ ] = {“thank} ;
 strcpy（s1 , s2）;
```

语句执行后，字符数组 s1 的结果为：“thank”

说明：

（1）字符数组 1 的长度要能够存放字符数组 2，否则出错。

（2）字符数组 1 必须是字符数组名，字符数组 2 可以是字符数组名，也可以是字符串常量。

（3）字符串复制到字符数组 1 后，字符数组 1 中只会覆盖字符数组中相应的元素（包括字符数组 2 中的结束标志 ’\0’），未覆盖的元素保留原有的数据。

（4）不能用赋值语句将字符串或字符数组直接赋值给另一个字符数组。例如：以下语句是错误的：

```
char s1 [ 5 ] , s2 [ 5 ] ;
s1 = “ abcd ” ;　　//字符数组名代表数组起始地址，不能将字符常量赋值给地址
s1 = s2 ;　　//字符数组不能直接复制给另一个字符数组
```

但是，可以将字符常量赋值给字符元素。例如，以下语句是正确的：

```
char s1 [ 5 ] , s2 [ 5 ] ;
s1 [ 0 ] = ‘ t ‘ ;　s2 [ 4 ] = ‘ w ‘ ;　s1 [ 3 ] = s2 [ 4 ] ;
```

（5）可以用 strncpy（）函数复制字符串 2 的前 n 个字符。但是 n

不能大于字符串 2 的个数。例如：

strncpy（s1 , s2 , 3）；　//将字符数组 s2 的前 3 个字符复制到字符数组 s1 中。

5. 字符串比较函数 strcmp（）

一般形式为：

strcmp（字符串 1，字符串 2）

比较字符串 1 和字符串 2，如果字符串 1>字符串 2，则函数值为正整数；如果字符串 1<字符串 2，则函数值为负整数；如果字符串 1 等于字符串 2，则函数值为 0。

以下语句都是正确的。

```
strcmp（s1 , s2）;
strcmp（”abcd” , “abcf”）;
strcmp（s1 , “abcd”）;
strcmp（”abcdefg” , s1）;
```

说明：

（1）字符串的比较规则是：将两个字符串从左到右依次逐个字符进行比较（按 ASCII 码表中的大小），直到出现不相同的字符或字符结束标志’\0’。

（2）如果所有字符都相同，则认为两个字符串相同。

（3）如果出现不相同的字符，则以第 1 对不相同的字符比较的结果为准。

（4）不能字符数组名进行比较。例如以下的代码是错误的：

```
if（s1 == s2 ）　　// s1 和 s2 都是数组名
    printf（”yes\n”）;
else
    printf（”no\n”）;
```

例 6.8 根据提示，猜国家的名称。

分析：根据屏幕的相关提示，输入一个国家的名称，判断是否输入正确。先将正确的国家名称保存在字符数组 c1 中，输入的国家名称保存在字符数组 c2 中，然后对 c1 和以 2 进行比较,相同就正确，不同就错误。

程序如下：

```
#include   <stdio.h>
#include   <string.h>
main（）
{
        char c1 [ ] = { “china”} ;
        char c2 [20] ;
        printf（”具有 5000 年文明;\n”）;
        printf（”龙的人\n”）;
        printf（”请输入国家的名称：”）;
        scanf（”%s ” , s2）;
         if（stcmp（c1 , c2）== 0）
             printf（”恭喜你回答正确！ \n”）;
         else
             printf（”不好意思，回答错误！ ”）;
}
```

6. 字符串长度函数 strlen（）

一般形式为：

strlen（字符数组）

返回字符数组（也可以是字符串）的实际长度，不包括结束标志’\0’。

例如：

char s1 [10] = { “china” } ;

printf（”%d” , strlen（s1））； //输出的值为 5

7. 大小写转换函数 strlwr（）和 strupr（）

一般形式为：

strlwr（字符串）　　//将字符串转换成小写字母

strupr（字符串）　　//将字符串转换成大写字母

6.3.7 字符数组程序实例

6.9 输入一个字符，判断该字符在某一个字符串的位置（第一次出现的位置），如果该字符没有在字符串中出现，就输出“no find”。

分析：定义一个字符串 s1 并初始化，再定义一个字符变量 c1，接收输入的字符，用一个循环，依次从字符串中截取一个字符进行比较，如果相同，则循环变量的值就是字符在字符串中出现的位置。

程序如下：

```
#include   <stdio.h>
#include   <string.h>
main（）
{
    char c1 , c2 , s1 [ 20 ] = { “hello wrold”} ;
    int n ;
    c1 = getchar（）;
      for（n = 0 ;（c2 = s1 [ n ]）!= ‘ \0 ‘ ; n++）
          if（c1 == c2）   //如果找到了就显示输出位置，并退出循环
                {
printf（”字符%c 在字符串%s 中出现的位置是：%d ” , c1 , s1 , n）;
break ;
```

```
}
        if（n == strlen（s1）） //如果循环到最后，表示没有找到
            printf（”no find”）;
}
```

项目小结

本项目主要讲解了数组的含义和作用，不同的数据类型保存到相对的数组类型中，以及如何用数组来编写相应的程序。通过学习，大家要掌握数组编程的优点。

习题

一、程序填空

1. 以下程序的功能是：输入 5 个数，求平均值。在空白处填上合适的代码。

```
#include   <stdio.h>
#include   <math.h>
int main（）
{
   float   a[5],_____;
   int    m;
   for（m=0 ; m<5; m++）
scanf（"%f",&a[m]）;
   for（m=0 ; m<5; m++）
```

```
    ____________________________________
    s/=5;
    printf（"这 5 个数的平均值为: %f",s）;
    return 0;
}
```

2. 以下程序的功能是：输入一个字符串，求其长度。在空白处填上合适的代码。

```
#include   <stdio.h>
int main（）
{
    int i=0j
    char str[30];
    printf（"输入字符串:"）;
    ____________________________
    while（_________________）
    i++;
    printf（"字符串的长度为:%d\n", i）;
    return 0;
}
```

二、编写以下程序

1. 求一个 3×3 矩阵对角线元素之和。

2. 从键盘输入 20 名学生的成绩，找出其中的最高分和最低分，并计算平均分。

3. 将字符数组元素倒序存储。

4. 有一个已经从小到大排序的整型数组，现在输入一个整数，要求按原来的规律将它插入到数组中。

5、将一个十进制整数转化为二进制整数并输出。

6、编写程序实现统计一个字符串中元音字母的个数。

项目七 函 数

通过前面的学习，我们知道，一个 C 程序必须有一个 main（）函数，称为主函数，程序要完成的功能，代码都放在主函数中。如果一个程序功能复杂，将所有代码全部放到主函数中，会使主函数结构不清晰，阅读和调试困难。另外，当有些功能需要重复使用时，代码就要重新编写，效率不高。

我们可以将一个复杂的程序，按功能分解成几个不同的小模块，每一个模块完成一个特定功能，然后，将这些模块“组装”成一个完整的程序，这就是模块化程序设计思想。

C 语言中，模块是由函数来实现的，

任务 7.1 函数概述

任务目标

1、理解函数的含义

2、了解函数的注意事项

任务实施

一个 C 程序由一个 main（）函数和若干个其它函数组成。主函数可以调用其它函数，其它函数之间也可以互相调用，但是不能调用主函数。函数可以被调用多次。

可以把函数理解成“功能”，一个函数完成一个特定的功能，所这些功能组合在一起，就组织成一个条理清晰、功能完整的 C 程序。

函数可以是系统提供的，称为库函数，这些函数保存在“头文件”中，用户可以直接使用，使用前需要将定义该函数的头文件用#include 语句包含进来。

函数也可以是用户自己定义的函数，该函数的定义和功能由用户编程实现。

使用函数，可以使代码利用率提高，对相同的功能，只需要调用函数就可以了。例如：一个班上有 50 名学生，每名学生有语文、数学、英语三门成绩，如果要计算每门课的平均分，我们不用分别给这三门课都来进行求平均分的计算，只需要编写一个求平均分的函数，然后，分别用语文、数学和英语成绩去调用求平均分函数就行了，这使得代码重复利用率大大提高。

同样，如果要输出相同的内容，可以将这部分内容提炼出来，放到一个函数里，就可以进行反复调用了。

例 7.1 输出以下结果。

```
************
hello    world
************
```

按照以前的知识，我们可以用以下代码实现：

```
#include   <stdio.h>
main（）
{
   int n ;
   for（n = 0 ; n< 12 ; n++）      //输出 12 个星号
      printf（” * “）;
   printf（”\nhello    world\n”）;
      for（n = 0 ; n< 12 ; n++）      //输出 12 个星号
         printf（” * “）;
}
```

从上面的例子中可以看出，第一行输出的 12 个星号和第三行输出的 12 个星号代码完全一样，但是出现了 2 次，这样使得 main（）函数中的代码冗长，而且代码维护不方便，如果想输出 15 个星号，则两处代码分别都要进行修改。将上面的代码用函数修改如下：

```
#include   <stdio.h>
main（）
{
    void outstar（）;     //声明输出星号的函数
    outstar（）;          //调用函数
    printf（”\nhello    world\n”）;
    outstar（）;          //调用函数
}
```

```
void outstar（）              //定义输出星号的函数
{
    for（n = 0 ; n< 12 ; n++）      //输出 12 个星号
        printf（” * “）;
}
```

将输出星号的代码放到 outstar（）函数里，使 main（）函数简洁，如果想输出 15 个星号，只需要在 outstar（）函数中修改一次即可。

说明：

（1）一个源文件由若干个函数组成，编译时，是以源文件为单位进行编译。

（2）主函数 main（）只能有一个，C 程序总是 main（）开始运行，main（）函数可以调用其他函数，其他函数不能调用 main（）函数，它是被操作系统调用的。

（3）其它函数是并列的，可以互相调用，而且可以多次调用。

（4）函数可以有参数，也可以没有参数。

（5）函数可以有返回值，也可以没有返回值。

任务 7.2 定义函数

任务目标

1、掌握函数的定义方法

2、了解函数的形式参数、实际参数和返回值

任务实施

以 C 语言中，函数要求“先定义后使用”，定义函数是为了在调用之前，告诉编译系统，程序里有一个什么样的函数，它有什么参数，有什么返回值，有什么功能等等。如果不定义函数，编译系统怎么知

道将来会有这样一个函数存在呢?所以不定义就调用这个函数的话，系统会报错。

7.2.1 函数定义的一般形式

返回值类型　函数名（参数列表）

{

　函数体；

}

例如上面例子中的 outstar（）函数：

```
void    outstar（）              //定义输出星号的函数
{
    for（n = 0 ; n< 12 ; n++）      //输出 12 个星号
        printf（” * “）;
}
```

void：返回值类型，表示没有返回值，此时可以省略；

outstar（）：函数名，以后就是用函数名进行调用。括号里是参数，如果什么也没有，表示没有参数。当然根据实际需要，参数可以有一个或多个；

花括号里的语句是函数体，用代码实现函数的功能。

例 7.2 定义一个求两个整数最大值的函数。

```
int    max（int a , int b）    //返回值为整型，必须和 return 语句的值类型一致
{
    int    temp ;
    temp = a > b ? a: b ;
    return（temp）;      //返回 temp 的值，括号不是必须的。
}
```

说明：

（1）定义函数时，要在主函数之外定义，一般放在主函数的后面。在主函数中，调用函数之前，要对函数进行声明。

（2）定义函数时，指定了两个整型变量 a 和 b ，这两个变量是形式参数，其值由主调函数的实际参数值传递而来。

（3）函数前面的 int，表示该函数有返回值，且返回值的类型是整型，该返回值由 return 语句来完成。

7.2.2 定义空函数

如果只有函数名，其它什么都没有，形如以下的格式：

函数名（）

{ }

则称该函数为空函数，它什么工作也不做，只占一个位置。通常用来进行系统功能扩展。在主函数中可以调用空函数。

任务 7.3 函数调用

任务目标

1、掌握函数调用的一般方法

2、掌握函数参数的传递方式

3、掌握函数编写程序的方法

任务实施

定义函数的目的是为了调用该函数，来获得预期的值。

7.3.1 函数调用的形式

在程序中，直接用函数名以及对应的参数进行调用，相当于告诉程序，在什么时候运行函数代码。如果是调用无参函数，直接用函数

名，如果是有参函数，则在函数名的括号中加上实际参数的值。

例如，调用前面定义的 outstar（）函数和 max（）函数，可以这样调用

```
outstar（）;
c = max（10 , 5）;
```

注意，调用函数时，函数名后在的括号不能省略。

一般来说，函数调用有三种形式：

（1）函数作为语句调用

这种调用方式适合函数没有返回值，只完成一定的操作。比如输出一个特定的值或符号等。前面的 outstar（）;就是这种调用。

（2）函数作为函数表达式调用

函数调用出现在另外的某个表达式中，作为表达式的一部分。这要求函数要有一个确定的返回值，来参与表达式的运算。如：上面的 c = max（10 , 5）; 它是将函数值赋值给变量 c，此时 max（）函数作为了赋值表达式的一部分。

（3）函数作为函数参数调用

函数调用出现在另一个函数的参数中。也要求函数有确定的返回值。

如：printf（”%d” , max（a , b））;

将函数 max（）作为 printf（）函数的参数进行调用。它相当于以下两名代码的功能

```
c = max（a , b）;
printf（”%d” , c）;
```

显然，作为参数进行调用，代码更简洁，效率更高。

7.3.2 函数调用时的数据传递

1. 形式参数和实际参数

定义函数时在括号内指明的参数，称为形式参数；调用函数时，在函数括号内指明的值称为实际参数。形式参数只能是变量，实际参数可以是常量，变量，表达式。注意，形式参数和实际参数在个数和数据类型上必须一一对应。

2. 实际参数和形式参数之间的数据传递

在函数调用时，将把实际参数的值传递给函数的形式参数，形式参数用获得的值在该函数中进行操作，直到该函数结束。

例 7.3 从键盘输入 2 个整数，输出较大的数。其中，求最大值的功能用函数来实现。

程序如下：

```
#include   <stdio.h>
main（）        //主函数
{
    int max（int   a , int   b）;      //声明求最大值函数
    int   int   x , y , z ;              //定义三个变量
    printf（”请输入两个整数：”）;        //输入提示
scanf（%d%d” , x , y）;              //输入两个整数
z = max（x , y）;
//调用 max（）函数，就最大值，并将最大值赋值给 z
printf（”较大的数是：%d\n” , z）;
return   z ;
}
int   max（int   a, int   b）         //定义求最大值函数
{
   int c;
```

```
    c = a > b ? a : b ;          //将最大值赋值给 c
    return   c ;                 //返回 c
}
```

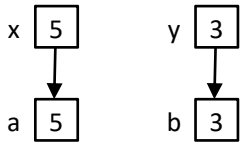
说明：

（1）在定义函数时，必须指定形式参数的类型。

（2）定义函数时，指定的形式参数只有在函数调用时才会分配内存单元，当该函数调用结束时，形式参数所占内存空间会被释放。

（3）实际参数和形式参数的类型和个数应能够完全匹配。

（4）参数传递时，只能由实际参数传递给形式参数，这个过程是单向的。从本例中，实际参数 x 和 y 的值从键盘输入获得（假设从输入的值为 5 和 3），然后调用 max（）函数时，将值传递给形式参数 a、b。注意，形式参数值的改变并不会影响实际参数的值。



7.3.3 对被调函数的说明

在一个函数中，调用另一个函数（即被调函数）需要具备以下条件：

（1）被调函数必须存在，可以是库函数或是自己定义的函数。

（2）如果是调用库函数，则在程序开始用#include 语句将相关库函数的信息包含到本文件中来。例如之前所用到的#include <stdio.h>，其中的“stdio.h”是一个头文件，即标准输入输出的意思，它包含了有关输入输出函数的信息，如果不将 stdio.h 头文件中的信息包含到进来，则程序中就无法使用相关的输入输出函数了。

（3）如果调用用户自己定义的函数，而且函数的定义在调用该函数的语句之后，则在主调函数中调用该函数之前应该出现对该函数的

声明。

函数声明的作用是将函数名，函数参数个数以及参数类型告诉编译系统，以便在调用函数时，编译系统能正确识别该函数。函数声明包括函数返回值，函数名，参数类型（可以没有参数名）和个数，没有函数体。

如果被调函数出现在调用函数之前，则可以不进行函数声明。

例 7.4 输入两个实数，用函数进行求和。

```
#include   <stdio.h>
main（）
{
      float   add2（float   f1 , float   f2）;    //声明函数
      float   x , y , z ;
      printf（”请输入两个实数：”）;
      scanf（”%f %f” , x , y）;
      z = add2（x , y）;
      printf（”两个实数这和为：%f” , z）;
      return   0 ;
}
float   add2（float   f1 , float   f2） //定义求和函数
{
    float   f3 ;
    f3 = f1 + f2 ;
    return   f3;
}
```

由于在主函数中调用 add2（）函数，而该函数的定义出现在主函数的后面，所以在主函数中对该函数进行了声明。如果将 add2（）

函数的定义放在主函数的前面，则在主函数中可以不用声明。

仔细观察，发现函数的声明和函数定义的首行基本一样，只是声明函数时，是一条语句，后面要加分号，函数定义时，首行不能加分号。函数定义的首行称为函数原型，其作用是调用函数时，进行合法性检查，当调用函数时，函数名，返回值类型，参数类型和个数不一致时，编译时会报错，从而保证能正确进行函数调用。

7.3.4 函数的返回值

如果希望函数调用能得到一个确切的值，这个值就是函数返回值。函数的返回值用 return 语句完成。关于函数的返回值，需要说明以下几点：

1. 一个函数中可以有多个 return 语句，程序执行到哪个 return 语句，哪个 return 语句就起作用。没有返回值，就可以没有 return 语句。

2. return （0）；和 return 0； 作用是一样的，这个括号可以不要。

3. return 语句的表达式值的类型应该和定义函数时指明的数据类型一致。

4. 如果没有函数返回值，则定义函数时，用关键字 void 说明，此时，函数体中不允许出现 return 语句。

5. 主函数 main（）中，也可以有 return 0 语句。返回值为 0，表示主函数正常运行。

任务 7.4 函数的嵌套调用

任务目标

1、理解函数嵌套调用的含义

2、理解函数嵌套调用的执行过程

任务实施

C 语言规定，函数的定义是独立、平行的，也就是指在定义函数时，一个函数内不能再定义函数。但函数可以嵌套调用，就是指 A 函数调用 B 函数，而 B 函数中又调用 C 函数，程序执行过程如图 7－1 所示：

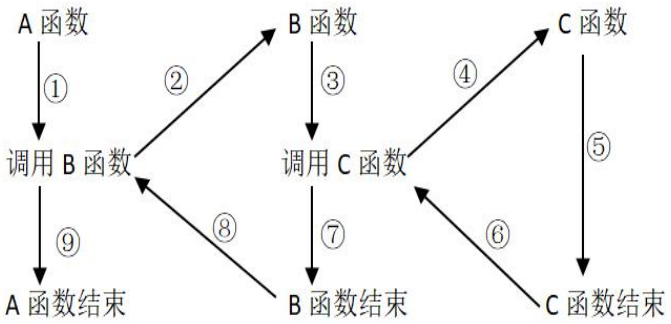


图 7－1 函数嵌套调用执行过程

①执行 A 函数的开始部分；②遇到调用 B 函数语句，则转去执行 B 函数；③执行 B 函数的开始部分；④遇到调用 C 函数语句，则转去执行 C 函数；⑤执行 C 函数，如果没有函数调用，则依次执行 C 函数直到结束；⑥C 函数执行完成，转回调用它的函数（即 B 函数）；⑦接着执行 B 函数中未执行的部分，直到 B 函数结束；⑧返回调用它的函数（即 A 函数）；⑨接着执行 A 函数中示执行的部分，直到 A 函数结束。

例 7.5 数组中有 10 个整数，求其中偶数的阶乘之和。

分析：分别编写判断偶数的函数和求阶乘的函数。主函数中依次传递数组中的元素去判断是否为偶数，如果是，则再调用阶乘函数。

程序如下：

```
#include   <stdio.h>
int   even（int   a）   //判断是否为偶数，是偶数返回值为 1
{
```

```
    if（a%2 == 0）
       return   1 ;
    else
       return   0 ;
}
float   fact（int   b）     //求阶乘
{
    float   s = 1.0 ;
    int   m ;
    for（m=1 ; m<=b;m++）
       s = s * m ;
    return   s ;
}
main（）
{
   int   num[10] = {1，2，3，4，5，6，7，8，9，10} ；
   int   n ;
   float   sum =0.0;    //保存阶乘和
   for（n = 0 ; n < 10 ; n++）
   {
       if（even（num [ n ]）== 1）  //是偶数就计算阶乘
           sum = sum + fact（num[ n ]）;
   }
   printf（"10 个数中的偶数的阶乘之和为：%f" , sum）;
   }
```

任务 7.5 数组作为函数参数

任务目标

1、理解数组作为函数参数的意义

2、掌握用数组作为函数参数编写程序的基本方法

任务实施

调用函数时，如果需要有参数，则必须调用时指定实际参数，实际参数可以是变量、常量、表达式等，还可以用数组来作为实际参数。用数组作为函数参数，可以是数组元素，也可以是数组名，它们表示的方法有很大的区别。

7.5.1 用数组元素作函数实际参数

数组元素相当于变量，可以作为函数的实际参数，但是不能作为函数的形式参数。因为形式参数是在函数被调用时临时分配的内存单元，而数组是一个整体，其数组元素是连续在内存中存放的。

在用数组元素作为函数实际参数时，把值单向传递给形式参数，是一种“值传递”方式，即把实际参数的值传递给形式参数，而形式参数的值不会传递给实际参数。

例 7.6 从包含 10 个元素的整型数组中找出最大的值（用函数实现）。

分析：主函数中，定义一个包含 10 个元素的数组 n，存放 10 个整型数；定义一个变量 m，存放当前的最大值，最开始时，假设数组第一个元素是最大值，赋值给 m；定义一个包含 2 个参数的函数 max，来求 2 个数的最大值。调用时，依次用 m 和数组的每一个元素进行比较，较大的就赋值给 m，比较完成，则 m 中的值就是取大值。

程序如下：

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

```
main（）
{
    int   max（int   a , int   b）;       //声明 max 函数
    int   n[10] , m , i ;
    printf（”请输入 10 个整数：”）;
for（i = 0 ; i < 10 ; i++）
    scanf（”%d” , &n[i]）;
m = n [ 0 ] ;
for（i = 1 ; i <10 ; i++）
//调用 max 函数，依次用 m 和数组每个元素比较，把较大的值赋给 m
m = max（m , n [ i ]）;    //变量 m 和数组元素作为实际参数
printf（”\n10 个数里最大的值是：%d” , m）;
return   0 ;
}
int   max（int   a , int   b）
{
  return   a > b ? a : b ;
}
```

运行结果：

```
请输入 10 个整数：10   5   8   7   23   65   19   54   63   12
10 个数里最大的值是：65
```

7.5.2 数组名作为函数参数

数组名可以作为函数参数，包括形式参数和实际参数。如果实际参数用数组名时，向形式参数传递的是数组首元素的地址。

例 7.7 有三个数组分别存放学生的语文、数学、英语成绩，用函数来计算各科的平均分。

分析：请平均分的代码都是一样的，没必要为语文、数学、英语三门课都来写一遍，只需要写一个求平均分的函数，每次调用时，用不同的数组名作为实际参数进行调用就行了。

程序如下：

```
#include   <stdio.h>
float   average（float   cj[ 10 ]）   //函数写在 main 函数前面，使用时不需要声明
{
    int   i ;
    float   aver , sum = 0 ;
    for（i = 0 ; i<10 ; i++）
          sum = sum + cj [ i ] ;
    aver = sum / 10 ;
    return   aver ;
}
main（）
{
float   yu[ 10 ] = {74,92,66,74.5,79,63,84,86,78,85.5};
float   shu[ 10 ] = {83.5,62,56,72,54,77,93,100,71,80};
float   yy[ 10 ] = {87,99,72,73,89.5,62,74.5,81,63,44};
   float   yuaver , shuaver , yyaver ;    //存放各科的平均分
   yuaver = average（yu）;          //数组名作为实际参数，计算语文平均分
   shuaver=average（shu）;        //数组名作为实际参数，计算数
```

学平均分

yyaver=average（yy）;　　//数组名作为实际参数，计算英语平均分

printf（”语文平均分是：%f\n”, yuaver）;

printf（”数学平均分是：%f\n”, shuaver）;

printf（”英语平均分是：%f\n”, yyaver）;

return　0;

}

说明：

（1）用数组名作函数参数，应该在主调函数和被调函数中分别定义数组，本例中，实际参数名分别是三个数组名 yu、shu 和 yy，形式参数名 cj 也是一个数组名。

（2）实际参数的数组名再调用函数时，是将数组的开始地址传递给形式参数的数组，此时，实际参数和形式参数的首地址都相同，也就是实参数组和形参数组使用相同的地址。如第一次调用 average 函数时，将数组 yu 作为实际参数，则实参数组和形参数组在内存中的使用情况下图所示：

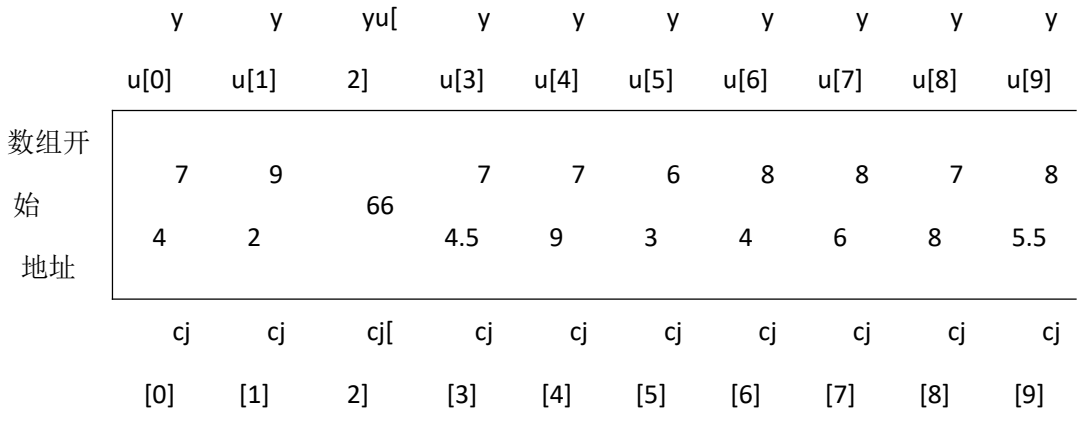


图 7－2 实参数组和形参数组在内存的使用情况

可以看出，yu[0]和 cj[0]，yu[1]和 cj[1]，……，yu[9]和 cj[9]分别使

用同一个内存地址，也就意味着改变形参数组里数组元素的值，实参数组对应的数组元素的值也会改变。

（3）形式参数数组与实际参数数组的类型应该一致，否则会报错。

（4）定义 average 函数时，指定形式参数数组的大小是不起作用的，编译系统并不检查形式参数数组的大小，只是将实际参数数组的第一个元素的地址传递给形式参数。此时，形式参数的第一个元素和实际参数的第一个元素是相同的地址。也就意味着形式参数数组和实际参数数组是共用相同的地址单元。因此，在定义函数时，形式参数数组只需要在数组名跟一个空的方括号就行了。如下：

```
float  average（float  cj [ ]）      //定义 average 函数，不需要指明形式参数的大小
```

为了程序便于阅读和检查，通常将形式参数数组的大小指定和实际参数大小相同。另外，可以在定义函数时，再指定一个参数，用了确定数组的大小，则程序更加灵活。

例 7.8 假设有三门课，参加考试的人数不相同，求平均值，如何用一个函数实现。

分析：由于传递参数时，并不检查形式参数数组的大小，因此，只需要在定义函数的时候，除了数组名作为参数外，再加一个参数，来确定数组的大小。假设语文有 6 个成绩，数学有 5 个成绩，英语有 10 个成绩，程序代码修改如下：

```
#include  <stdio.h>
float  average（float  cj[ ] ,  int  n）  //变量 n 用来确定数组大小
{
    int  i ;
    float  aver , sum = 0 ;
```

```
        for（i = 0 ; i < n ; i++）
              sum = sum + cj [ i ] ;
        aver = sum / n ;
        return    aver ;
    }
    main（）
    {
    float    yu[ 10 ] = {74,92,66,74.5,79,63};
    float    shu[ 10 ] = {83.5,62,56,72,54};
    float    yy[ 10 ] = {87,99,72,73,89.5,62,74.5,81,63,44};
       float    yuaver , shuaver , yyaver ;     //存放各科的平均分
       yuaver = average（yu , 6）;          //数组名作为实际参数，计算语文平均分
       shuaver=average（shu , 5）;        //数组名作为实际参数，计算数学平均分
       yyaver=average（yy , 10）;          //数组名作为实际参数，计算英语平均分
       printf（”语文平均分是：%f\n” , yuaver）;
    printf（”数学平均分是：%f\n” , shuaver）;
    printf（”英语平均分是：%f\n” , yyaver）;
    return    0 ;
    }
```

定义的 average 函数，其中的形式参数数组 cj，接收从主调函数传来的实参数组 yu 的首地址，并接收一个数组元素个数 6，则认为该数组就有 6 个元素；下一次调用时，用数组 shu 和数组元素个数 5 作为参数，表示该数组有 5 个数组元素。

再强调一次，形式参数数组接收的是实际参数数组的首地址，使用多少个数组元素，形式参数数组本身并不能确定，需要用户指定，在调用函数时，再用一个参数来指定数组大小是一个不错的方法。

任务 7.6 变量作用域

任务目标

1、理解变量作用范围的含义

2、掌握各种变量的使用方法

任务实施

每个变量的作用范围，称为变量的作用域。定义的变量，在作用域范围内有效，超出作用域范围，该变量无效。

定义变量有三种情况：在函数内部定义，在复合语句中定义，在函数外部定义。

7.6.1 局部变量

在函数或复合语句中定义的变量称为局部变量，它只能在定义变量的函数或复合语句中使用（先定义后使用）。超出这个范围，该变量不能使用。

例如：

```
char   fun1（char   c）
{
    char   c1;
    …
}
```

字符变量 c 和 c1 都是局部变量，只在函数 fun1 中有效。

```
int   fun2（int   a , int   b）
{
    int   x , y ;
    …
}
```

整型变量 a、b 和 x、y 是局部变量，只在函数 fun2 中有效。

```
main（）
{
    int   m , c ;
    {
        int   p , q ;
        … …
    }
… …
}
```

p、q 只在花括号里有效。

m、c 在主函数里有效。

说明：

（1）主函数中的定义的变量也是局部变量，只能在主函数中使用。上面的例子中 m 和 n 是局部变量，只在 main（）函数里有效。

（2）形式参数也是局部变量。

（3）不能函数中可以使用相同的变量名，但是表示两个不同的变

量。上例中，fun1 函数中有一个形式参数 c，main 函数中也定义了一个局部变量 c，它们表示两个不同的变量，在对应的函数中使用，不会互相干挠。

（4）定义局部变量的函数结束，局部变量所占用的内存空间将会被释放。

7.6.2 全局变量

在函数之外定义的变量称为全局变量，它的作用范围是从定义该变量的位置开始到本源文件结束，其后面所有函数均可以使用全局变量。

请看如下程序：

```
float   f1 = 3 , f2 = 5 ;        //全局变量
char   fun1（char   c）
{
    char   c1;
    …
}
int   fun2（int   a , int   b）
{
    float   x ;
    x = f1 + f2 ;   //函数中使用全局变量
}
int   m1 , m2 ;    //全局变量
main（）
{
    int   m , c ;
    {
        int   p , q ;
        … …
    }
… …
}
```

f1、f2 的作用范围

m1、m2 的作用范围

说明：

（1）f1、f2、m1、m2 都是全局变量，但是作用范围不一样。其作用范围是从定义变量开始到源文件结束。

（2）如果在同一个文件中，全局变量和局部变量重名，则在局部变量作用范围内，全局变量不起作用。当然在程序设计时，就避免出

现这种情况，以免影响程序阅读。如：

```
int   a = 5 ; b = 10 ;    //a 和 b 是全局变量
int   add（int   a , int   b）   //a 和 b 是局部变量，在这个函数中，局部变量起作用
{
   return   a + b ;
}
main（）
{
   int   m , n ;
   m = 1 , n = 2 ;
   printf（”%d” , add（m , n））;
}
```

（3）通常情况下，一个函数只能有一个返回值，如果希望函数有多个返回值，则可以用全局变量来实现，因为全局变量在所有函数中都有效，在一个函数中改变了全局变量的值，在另一个函数中，其值也会变化。

例 7.9 编写一个函数，求温度的最大值、最小值和平均值。

分析：一个函数不能返回三个值，可以定义三个全局变量 tmax，tmin 和 taver，分别保存温度的最大值、最小值和平均值。

程序如下：

```
#include   <stdio.h>
float   tmax = 0.0 , tmin = 0.0 , taver = 0.0 ;    //定义三个全局变量
void   tcacl（float   tt[ ] , int   n）
{
    int   i ;
```

```
    float   tsum ;      //保存温度之和
    tmax = tt [ 0 ] ;
    tmin = tt [ 0 ] ;
    tsum = tt [ 0 ] ;
    for（i = 1 ; i < n ; i++）
    {
        if（tmax < tt [ i ]） tmax = tt [ i ] ;
        if（tmin > tt [ i ]） tmin = tt [ i ] ;
        tsum = tsum + tt [ i ] ;
    }
    taver = tsum / n ;
}
main（）
{
    float    t[10]={12.2,13.2,11.7,16.9,20.1,22.3,18.6,17.0,20.1,12.4};
//假设有 10 个温度值
    tcacl（t , 10）;   //调用函数，传递温度数组，共 10 个元素
    printf（"温度最大值为：%8.2f，最小值为：%8.2f，平均值为：%8.2f" , tmax , tmin , taver）;
    return   0;
}
```

虽然使用全局变量，可以减少函数参数的传递，也可以使函数的返回值增加，但是不提倡使用全局变量。因为使用全局变量，会降低程序的可读性，同时，由于全局变量在各个函数中都能使用，也会降低模块的可靠性和通用性。

项目小结

本项目学习了函数的相关知识，结构化程序设计要将各种不同功能的模块放在不同的函数中，从而使用程序条理清晰，逻辑明确，增强阅读性。

函数在使用过程中，要遵守先定义后使用的原则，一般说来，函数定义在后面，则在调用之前，必须对函数进行声明。定义函数时，要理解函数的形式参数、实际参数的含义和使用方法，要理解函数的返回值类型。掌握用函数编写程序的一般方法。

习题

7.1 分别用函数求最大公约数和最小公倍数。

7.2 用函数实现 33 矩阵的转置。

7.3 用函数实现十进制数转换成二进制数。

7.4 编写判断是否为素数的函数。

7.5 编写是否为偶数的函数。

7.6 编写求累加和的函数。

项目八 复合结构

在前面章节中介绍了 C 语言中的多种数据类型，例如：整型、字符型、浮点型、数组、指针……等等。但是在实际开发中，只有这些数据类型是不够的，难以胜任复杂的程序设计。例如：在员工信息管理系统中，员工的信息就是一类复杂的数据。每条记录中都包括员工的姓名、性别、年龄、工号、工资等信息。姓名为字符数组、性别为字符、年龄为整型、工号为整型、工资为整型。

对于这类数据，显然不能使用数组存储，因为数组各个元素的类型都是相同的。为了解决这个问题，C 语言中提供了一种组合数据类型“结构体”。

任务 8.1　结构体类型

任务目标

1、掌握定义结构体变量的方法

2、结构体变量的使用方法

任务实施

8.1.1 结构体概述

结构体是一种组合数据类型，由用户自己定义。结构体类型中的元素既可以是基本数据类型，也可以结构体类型。

定义结构体类型的一般格式为：

```
struct 结构体名
{
成员列表
};
```

成员列表由多个成员组成，每个成员都必须作类型声明，成员声明格式为： 数据类型 成员名;

下面来看一个具体的例子：

```
struct  Employee
{
char    name[8];
int age;
int id;
int salary;
};
```

这段代码中 struct 是关键字，Employee 结构体名，struct Employee 表示一种结构体类型。该结构体中有 4 个成员，分别为

name、age、id、salary，使用这种结构体类型就可以表示员工的基本信息。

8.1.2 定义结构体类型变量的方法

在 C 语言中，常用的定义结构体变量的方式有两种

1、先定义结构体类型，再定义结构体变量，一般形式为：

```
struct 结构体名
{
成员列表
};
struct 结构体名 变量名;
```

例如：

```
struct  Employee
{
char    name[8];
int age;
int id;
int salary;
};
struct Employee emp;
```

这种方式和基本类型的变量定义方式相同，其中 struct Employee 是结构体类型名，emp 是结构体变量名。

2、在定义结构体类型的同时定义变量，一般形式为：

```
struct 结构体名
{
成员列表
}变量名;
```

例如：

```
struct  Employee
{
char    name[8];
int age;
int id;
int salary;
emp,emp1,emp2;
}
```

这种方式将结构体类型定义与变量定义放在一起，可以直接看到结构体的内部结构，比较直观。

8.1.3 结构体变量的初始化

在 C 语言中，结构体变量初始化，本质上是对结构体变量中的成员进行初始化，使用花括号{ }在初始化列表中对结构体变量中各个成员进行初始化。

例如：

```
struct Employee emp={“rupeng”,20,1,10000}
```

或

```
struct  Employee
{
Char name[8];
int age;
int id;
int salary;
}emp={“rupeng”,20,1,10000};
```

编译器会将“rupeng”、20、1、10000 按照顺序依次赋值给结构体变量 emp 中的成员 name、age、id、salary。

任务 8.2　共用体类型

任务目标

1、掌握定义共用体变量的方法

2、共用体变量的引用方法

3、共用体类型的数据特点

任务实施

8.2.1 共用体概述

在 C 语言中,允许几种不同类型的变量存放到同一段内存单元中,这种几个不同的变量共同占用一段内存的结构,被称为共用体类型结构,简称共用体。

结构体和共用体的区别在于：结构体的各个成员会占用不同的内存，互相之间没有影响；而共用体的所有成员占用同一段内存，修改一个成员会影响其余所有成员。

共用体一般定义形式为:

union 共用体名

{

数据类型 成员名 1;

数据类型 成员名 2;

.....

数据类型 成员名 n;

}变量名表列;

说明：union 为共用体类型关键字，大括号中定义了共用体类型的

成员项，每个成员项由数据类型和成员名组成。

例如：下面这个实例是先定义共同体，再创建变量。

```
union data        //声明共用体类型
{
int i;
char ch;
double f;
};
union data a，b，c;   //用共用体类型定义变量
```

也可以在定义共用体的同时创建变量，如下：

```
union data
  {
  int i;
  char ch;
  double f;
  } a,b,C;
```

如果不再定义新的变量，也可以将共用体的名字省略：

```
union
{
int i;
char ch;
double f;
} a,b,c
```

结构体变量所占内存长度是各成员占的内存长度之和，而共用体变量所占的内存长度等于最长成员的长度。上面的共用体 data 中，成员 f 占用的内存最多，为 8 个字节，所以 data 类型的变量（也

就是 a、b、c）也占用 8 个字节的内存。

8.2.2 定义共用体类型变量的方法

只有先定义了共用体变量,才能在后续的程序中引用它。不能直接引用共用体变量,而只能引用共用体变量中的成员。

引用方法如下:

共用体变量名.成员名

例如：前面定义了 a,b,c 为共用体变量，下面的引用方式是正确的：

a. i（引用共用体变量中整形变量 i）

a. ch（引用共用体变量中字符变量 ch）

a. f（引用共用体变量中实型变量 f）

不能只引用共用体变量，例如下面引用是错误的：

Printf(“%”,a)

应该写成：Printf(“%”,a.i)

8.2.3 共用体类型数据特点

(1)同一个内存段可以用来存放几种不同类型的成员,但是在每一瞬间只能存放其中的一种,而不是同时存放几种。所以在某一时刻,共用体变的内存地址中存放的、起作用的是最后一次存入的成员值。

a.ch=‘a’;

a.f=1.5;

a.i=40;

在完成以上 3 个赋值运算以后，变量存储单元存放的最后存入的 40，原来的‘a’和 1.5 都被覆盖了。

(2) 在共用体变量的定义时，只能对其中一个成员的类型值进行初始化

union date

```
{
int i;
char ch;
double f;
} a={1,‘a’,1.5};    // 不能初始化 3 个成员，它们占同一段存储单元
union data a={16}; // 正确，对第 1 个成员初始化
```

(3)共同体变量的地址和它的各成员的地址都是同一地址。

例如：&a.i, &a.c, &a.f 都是同一值

任务 8.3　枚举类型

任务目标

掌握枚举类型数据的使用方法

任务实施

在实际问题中，有些变量的取值被限定在一个有限的范围内。例如，一个星期内只有七天，一年只有十二个月，一个班每周有六门课程等等。如果把这些量说明为整型，字符型或其它类型显然是不妥当的。为此，C 语言提供了一种称为“枚举”的类型。在“枚举”类型的定义中列举出所有可能的取值，被说明为该“枚举”类型的变量取值不能超过定义的范围。

枚举类型的定义形式为：

```
enum  枚举类型 {枚举值列表};
```

例如，我们列出一个星期有几天：

```
enum week
{ Mon,Tues,Wed,Thurs,Fri,Sat,Sun };
```

可以看到，我们仅仅给出了名字，却没有给出名字对应的值,这是因为枚举值默认从 0 开始,往后逐个加 1 (递增) ;也就是说，week 中的 Mon、Tues .... Sun 对应的值分别为 0、1 6

我们也可以给每个名字都指定一个值:

```
enum week
{Mon=1,Tues=2,Wed=3,Thurs=4,Fri=5, Sat=6; Sun=7}
```

更为简单的方法是只给第一个名字指定值：

```
enum week
{Mon=1,Tues,Wed,Thurs,Fri, Sat; Sun}
```

这样枚举值就从 1 开始递增，跟上面的写法是等效的。

定义枚举类型的好处是可以让编程者方便的使用自定义的变量值来替代数字值，这样会使我们的代码有更高的可读性，而从本质上讲枚举除了在代码中是以文字形式出现的变量值之外（内部还是数字）与整数值其实并没有什么区别。

任务 8.4　typedef 使用

任务目标

掌握 typedef 的使用方法

任务实施

C 语言允许用户使用 typedef 关键字来定义自己习惯的数据类型名称，来替代系统默认的基本类型名称、数组类型名称、指针类型名称与用户自定义的结构型名称、共用型名称、枚举型名称。一旦用户在程序中定义了自己的数据类型名称,就可以在该程序中用自己的数据类型名称来定义变量的类型、数组的类型、指针变量的类型与函数的类型等。

8.4.1 简单地用一个新的类型名称代替原来的类型名

例如：

typedef int Integer;　定 Integer 为类型名，作用与 int 相同

typedef float Real;　// 指定用 Real 为类型名，作用与 float 相同

Real i ,j ; // 用 Real 定义变量 i 和 j，相当于 float i, j;

上面的实例中，将变量 i ,j 定义为 Real，而 Real 等价于 float,因此 i ,j 是 float 型。

8.4.2 命名一个简单的类型名代替复杂的类型表示方法

例如：命名一个新的类型代替数组类型

typedef int num[5];　// 声明 num 为整型数组类型名

Num a;　// 定义 a 为整型数组名，它有 5 个元素

归纳起来，声明一个新的类型名的方法是：

1. 先按定义变量的方法写出定义体。
2. 将变量名换成新的类型名。
3. 在最前面加 typedef。
4. 然后可以用新类型名去定义变量。

项目小结

本项目主要介绍了几种构造数据类型，包括结构体、共用体和枚举类型。通过本项目的学习，大家应重点掌握以下内容：

1.结构体类型的声明、定义、初始化和引用的方法。

2.了解共用体类型、定义、初始化和引用的方法。

3.掌握 typedef 的使用方法。

习题

一、选择题

1.下列关于结构体的说法错误的是（　　）

A.结构体是用户自己定义的一种数据类型

B.在定义结构体数据类型时，可以为成员设置默认值

C.结构体中可设定若干个不同数据类型的成员

D.结构体中成员的数据类型可以是结构体

2.C 语言中，结构体类型变量在程序执行期间（　　）

A.所有成员一直驻留在内存中　　B.只有一个成员驻留在内存中

C.部分成员驻留在内存中　　D.没有成员驻留在内存中

3.有以下定义

```
union data
{
   int da1;
   float da2;
}demo;
```

则下面叙述中错误的是（　　）

A.变量 demo 与成员 da2 所占内存字节数相同

B.变量 demo 中各成员的地址相同

C.变量 demo 和各成员的地址相同

D.若给 demo.da1 赋值 90 后，demo.da2 中的值是 90.0

二、编程题

1.利用结构体数据类型编程实现得数的加法和减法运算。

2.有 5 名学生，每名学生的信息包括学号、数学成绩、英语成绩，从键盘输入 5 名学生的信息，要求使用数组存储信息，实现：

（1）输出成绩中有 100 分的学生的所有信息；

（2）求该 5 名学生 2 门课程的总平均分。

项目九 指针

针是 C 语言中广泛使用的一种数据类型。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； 并能象汇编语言一样处理内存地址，从而编出精练而高效的程序。指针极大地丰富了 C 语言的功能。 学习指针是学习 C 语言中最重要的一环， 能否正确理解和使用指针是我们是否掌握 C 语言的一个标志。同时， 指针也是 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，上机调试。只要作到这些，指针也是不难掌握的。

任务 9.1 关于指针

任务目标：

1.理解指针的概念

2.掌握指针在存储方式

3.掌握指针变量的定义

任务实施

9.1.1 指针是什么

C 语言里，变量存放在内存中，而内存其实就是一组有序字节组成的数组，每个字节有唯一的内存地址。CPU 通过内存寻址对存储在内存中的某个指定数据对象的地址进行定位。这里，数据对象是指存储在内存中的一个指定数据类型的数值或字符串，它们都有一个自己的地址，而指针便是保存这个地址的变量。也就是说：指针是一种保存变量地址的变量。

前面已经提到内存其实就是一组有序字节组成的数组，数组中，每个字节大大小固定，都是 8bit。对这些连续的字节从 0 开始进行编号，每个字节都有唯一的一个编号，这个编号就是内存地址。示意如下图：

这是一个内存模型。左侧的连续的十六进制编号就是内存地址，每个内存地址对应一个字节的内存空间。而指针变量保存的就是这个内存地址。

9.1.2.指针变量的定义

指针其实就是一个变量，指针的声明方式与一般的变量声明方式没太大区别，定义指针变量的格式是：

类型名　*指针变量名;

例如：

```
int *pointer1;
```

指针的声明比普通变量的声明多了一个一元运算符“*”。运算符“*”是间接寻址或者间接引用运算符。当它作用于指针时，将访问指针所指向的对象。在上述的声明中：pointer1 是一个指针，保存着一个地址，该地址指向内存中的一个变量；*pointer1 则会访问这个地址所指向的变量。

可以在定义指针变量时，进行初始化，如：

```
int a;
int *pointer1=&a;
```

以上代码的含义是变量 pointer1 指针指向 a。

如果一个指针没有被初始化，那么程序就不知道它指向哪里。它可能指向一个非法地址，这时，程序会报错，这种情况更严重，你的程序或许能正常运行，但是这个没有被初始化的指针所指向的那个位置的值将会被修改，而你并无意去修改它。用一个例子简单的演示一下：

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

```
{
    int *p;
    *p = 1;          // 错误：指针未初始化
    printf("%d\n",*p);
  system("pause");
}
```

要想使这个程序运行起来，需要先对指针 p 进行初始化：

```
#include <stdio.h>
#include <stdlib.h>
void main()
{
    int x=1;
    int *p=&x;
    *p = 1;
    printf("%d\n",*p);
system("pause");
}
```

任务 9.2 指针的使用

任务目标：

1、了解指针函数的使用

2、了解指针与数组的使用

3、了解指向函数的指针

任务实施

9.2.1. 指针与函数

函数的参数不仅可以是整型、浮点型、字符型等数据，还可以是指针类型。它的作用是将一个变量的地址传送到另一个函数中。

指针作为函数参数，使得被调函数能够访问和修改主调函数中对象的值。

下面通过一个例子来说明。

```
#include <stdio.h>
#include <stdlib.h>
void swap(int *a,int *b)    // 定义函数
{
    int temp;
    temp = *a;
    *a = *b;
    *b = temp;
}
void main()
{
    int x = 1,y = 2;
    swap(&x,&y);              // 将 x,y 的地址作为参数传递给了被调函数，传递过去的也是一个值，与传值调用不冲突
    printf("%d %5d\n",x,y); // 输出结果为：2      1
system("pause");
}
```

9.2.2. 指针与数组

在 C 语言中，指针与数组之间的关系十分密切。实际上，许多可以用数组完成的工作都可以使用指针来完成。一般来说，用指针编写

的程序比用数组编写的程序执行速度快，但另一方面，用指针实现的程序理解起来稍微困难一些。

一个变量有地址，一个数组包含多个元素，每个数组元素都在内存中占用存储单元，它们都有相应的地址。指针变量既然可以指向变量，当然也可以指向数组元素。所谓数组元素的指针就是数组元素的地址。

如以下程序：

```
# include <stdio.h>
void main(void)
{
int a[] = {1, 2, 3, 4, 5};
int *p = &a[0];
}
```

P 的值是数组 a 首元素 a[0]的地址。若执行 p++则表示指向 a[1]的地址。

在指针指向一个数组元素时，可以对指针进行以下运算：p++、p--分别表示同一数组的下一个元素和上一个元素。

```
#include <stdio.h>
#include <stdlib.h>
void main()
{
    int x[10] = {1,2,3,4,5,6,7,8,9,0};
    int *p = x;
    printf("x 的地址为：%p\n",x);
    printf("x[0]的地址为：%p\n",&x[0]);
    printf("p 的地址为：%p\n",&p);
```

```
        p += 2;
        printf("*(p+2)的值为：%d\n",*p);
    system("pause");
    }
```

输出结果如下：

x 的地址为：0x7FFE990B39F0

x[0]的地址为：0x7FFE990B39F0

p 的地址为：0x7ffe990b39e8

*(p+2)的值为：3

可以看到， x 的值与 x[0]的地址是一样的，也就是说数组名即为数组中第 1 个元素的地址。实际上，打印&x 后发现，x 的地址也是这个值。而 x 的地址与指针变量 p 的地址是不一样的。故而数组和指针并不能完全等价。

9.2.3. 指向函数的指针

一个函数总是占用一段连续的内存区域，函数名在表达式中有时也会被转换为该函数所在内存区域的首地址，这和数组名非常类似。我们可以把函数的这个首地址（或称入口地址）赋予一个指针变量，使指针变量指向函数所在的内存区域，然后通过指针变量就可以找到并调用该函数。这种指针就是函数指针。声明一个函数指针的方法如下：

返回值类型 （* 指针变量名）([形参列表])；

```
int (*pointer)(int *,int *);              // 声明一个函数指针
```

通过一个简单的案例实现函数的指针调用

```
#include <stdio.h>
//返回两个数中较大的一个
```

```
int max(int a, int b){
    if(a>=b)
        return a;
        else
            return b;
}
//返回两个数中较小的一个
int max(int a, int b){
    if(a>=b)
        return b;
        else
            return a;
}
void main(){
 int x, y, maxval,minval;
 //定义函数指针
 int (*p)(int, int);
 printf("请输入两个数:");
 scanf("%d %d", &x, &y);
p=max;
 maxval = (*p)(x, y);
p=min;
minval = (*p)(x, y);
 printf("大数: %d；小数:%d\n ", maxval, minval);
}
```

运行结果

请输入两个数：100,99

大数: 100；小数:99

9.2.4. 返回指针的函数

一个函数可以返回一个整型值、字符值等，也可以返回指针型的数据，即地址。其概念与以前类似，只是返回的值的类型是指针类型而已。

返回指针的函数的一般形式为：类型名 * 函数名(参数列表)

下面的例子定义了一个函数 strlong()，用来返回两个字符串中较长的一个：

```
#include <stdio.h>
#include <string.h>
char *strlong(char *str1, char *str2){
    if(strlen(str1) >= strlen(str2)){
        return str1;
    }else{
        return str2;
    }
}
int main(){
    char str1[30], str2[30], *str;
    gets(str1);
    gets(str2);
    str = strlong(str1, str2);
    printf("Longer string: %s\n", str);
    return 0;
}
```

运行结果：

```
C Language↙
c.biancheng.net↙
Longer string: c.biancheng.net
```

用指针作为函数返回值时需要注意的一点是，函数运行结束后会销毁在它内部定义的所有局部数据，包括局部变量、局部数组和形式参数，函数返回的指针请尽量不要指向这些数据，C 语言没有任何机制来保证这些数据会一直有效，它们在后续使用过程中可能会引发运行时错误。请看下面的例子：

```
#include <stdio.h>
int *func(){
    int n = 100;
    return &n;
}
int main(){
    int *p = func();
    //当前函数调用结束，函数内部变量即刻销毁，产生逻辑错误
    int n;
    n = *p;
    printf("value = %d\n", n);
    return 0;
```

项目小结

本项目主要讲解了 C 语言中的指针的用法。包括指针的概念、定义，用法。指针理解起来比较困难，这个项目可以作为中职阶段的选修内容，中职阶段的学生可以只了解指针的基本概念，对指针有个认识就行了。

习题

一、选择题

1.若有定义“int x=0, *p=&x;”，则语句“printf（"%d\n"，*p）;”的输出结果是（　　）

A.随机值　　B.0　　c.x 的地址　　D.p 的地址

2.若有定义语句“int *p,a;”，则语句“p=&a;”中的运算符“&”的含义是（　　）

A.位与运算　　B.逻辑与运算　　C.取指针内容　　D.取变量地址

3.下面程序中调用 scanf（）函数给变量 a 输入数值的方法是错误的，其错误的原因是（　　）

```
#include <stdio.h>
int main（）
{
   int *p,q,a,b;
   p=&a;
   scanf（"%d" , *p）;
   … …
}
```

A. *p 表示的是指针变量 p 的地址

B.*p 表示的是变量 a 的值 ，而不是变量 a 的地址

C.*p 表示的是指针变量 p 的值

D.*p 只能用来说明 p 是一个指针变量

二、填空题

1.定义 p1 为指向整形数据的指针变量_____

2.定义指针数组 p2，它由 4 个指向整形数据的指针元素组成_____

3.定义 p 为指向函数的指针，该函数返回一个整形值_____

三、编程题（全部用指针实现）

1.输入 3 个整数，按由大到小的顺序输出

2.编写程序，实现逆序存放数组元素的值。

3.将字符串按由小到大的顺序排列。

参考文献

[1]谭浩强 C 程序设计（第五版）[M].北京：清华大学出版社，2017

[2]王剑峰 C 语言程序教程 北京：航空工业出版社，2020

[3]郝贤云 C 语言程序设计 上海：同济大学出版社，2019

C语言程序设计

主　编：杨从林
副主编：龙敏　王逊
参　编：苏鹏权　胡钢