C 语言入门代码书

起稿者：温宇彤

2015 年 10 月 7 日

目录

摘要：C 语言是世界上最受欢迎的语言之一。
关键字：C 语言、程序设计、入门教程

第一部分：概述

一、 C 语言概述

1. 什么是程序设计语言

一台没有安装任何软件的计算机称为“裸机”，只有借助各种软件，计算机才能进行各项工作。软件由程序和程序的相关文档构成，程序是软件的核心部分，由具体的程序设计语言编写。

上世纪五十年代，早期的计算机使用汇编语言和机器码来编写程序，它们编写的语句很难记忆，而且非专业人员很难掌握。于是很多研究人员开始致力于高级语言的开发。经历过 Fortain（1957）、Basic（1964）等语言之后，美国贝尔实验室在 1972 年设计出 C 语言。在 1973 年，研究人员用 C 语言编写了 UNIX 系统 90%以上的部分，并发表了 UNIX 5 版本。1978 年以后，C 语言被广泛应用在各类计算机的程序设计上，并且还用来编写操作系统，成为当代最优秀的程序设计语言之一。

随着计算机的普及，出现了许多 C 语言的版本，由于没有统一的标准，使得众多种类的 C 语言有了一些不一致的地方。为了改变这种现状，美国国家标准研究所（ANSI）制定了一套 C 语言的国际标准，成为现行的 C 语言标准。

2. C 语言的特点

2.1. 语言简介、紧凑、灵活

C 语言有 32 个关键字，9 种控制结构，程序书写灵活自由。

2.2. 运算符丰富

C 语言中有 34 种运算符，其中包括一些具有 C 语言特性的运算符，使 C 语言能更容易实现算法。

2.3. 功能丰富

C 语言有各种数据类型，能够实现各种复杂的数据结构。引入指针概念，配合位运算，可以直接访问物理地址，能对硬件直接进行操作，虽然 C 语言是一种高级语言，但同时也具备低级语言的大部分功能。

2.4. 结构化程序设计语言

函数是 C 程序的基本模块，每个函数实现一个程序中的一个功能。一个程序分成多个函数模块，每个模块可以分开调用，也可以分模块对程序进行修改。C 语言同时具有多种程序控制结构，使程序完全结构化。

2.5. 可移植性强

C 语言是一种可移植语言，在一个系统上编写的 C 程序可以在其他系统上运行。和其他高级语言相比，C 语言在可移植性方面处于领先地位。

2.6. 高效

C 语言的运行效率在高级语言中仅次于汇编语言，在其他高级语言中运行效率非常出众。

3. 程序设计初步

3.1. 程序设计的一般步骤

一般人们会认为，程序设计就是编写代码到运行成功的过程。这是一个误区。程序设计大概由以下的步骤组成。

a. 分析

在这一步骤中，要明确要解决的问题。通过对问题的分析，明确要写的程序具有什么样的功能。不需要考虑如何完成程序。这一阶段也成为需求分析。

b. 程序设计

在设计阶段要考虑到程序界面、算法、操作、数据结构等的设计问题。当处理复杂的问题的时候，选择合适的数据结构会提高程序运行的效率。这个阶段是对程序整体的规划，不涉及代码。

c. 编写代码

对程序进行分析和设计之后，就要把自己构思好的程序通过 C 语言知识编写成程序代码。存放程序代码的文件叫做源程序。

d. 编译、连接、运行

编写好的源程序在编译器中进行编译，生成二进制代码（计算机唯一能识别的机器码），如果程序有错误，会在这个阶段生成提示信息，用户经过修改之后可以再次编译。编译通过之后会生成以.obj 为后缀的二进制代码文件。将得到的二进制代码文件与系统标准模块（如 C 语言的标准函数库）经过连接处理之后，获得后缀名为.exe 的可执行文件，才能够在计算机中运行。

e. 测试与调试

某些情况下，运行成功的程序不一定有正确的运行结果，这是由于程序中会有某些逻辑错误（bug）。这个时候就需要对程序进行测试和调试。测试与调试的工作往往比编写程序更加花费时间。

3.2. 编写第一个 C 语言程序

```
1. //第一个 C 语言程序
2. #include<stdio.h>
3. int main( )
4. {
5.         int    n;
6.         n = 1;
7.         printf(“This is my NO.%d program” , n);
8.         return    0;
9. }
```

程序分析：

a. 第 1 行位于两个斜线之后的字符成为注释，对程序有解释的功能。注释在小型程序中没有太大显著的作用，往往在更加大型的程序程序愈加显著。注释有两种形式，一种是以两个斜线“//”开头的行注释，在两个斜线之后的一行代码都会无效。另一种形式的注释是由“/*”和“*/”一对符号组成的块注释，从一个“/*”开始，一直到下一个“*/”出现之前的所有代码都会无效。
b. 第 2 行为预编译指令，stdio.h 文件中包含着 C 语言中有关于输入输出功能的预定义，在这里需要用到对应功能，所以要把这个文件包含进来。这个文件被称作“头文件”。
c. 第 3 行为程序的主函数。一个 C 程序有且只能有一个主函数，C 程序从主函数里的第一行开始运行。主函数规定的返回值在不同教材中规则不一，在这里统一规定主函数的返回值是整形。并且在主函数的最后一行中要让 main 函数返回 0（第 8 行）。有关于返回值的相关内容将在函数一章学习，在这里可以大致理解为返回值就是函数值。
d. 第 5 行定义了一个整形变量 n，第 6 行给 n 赋值为 1，第七行输出一行字：

```
This is my NO.1 program.
```

3.3. 编程环境

C 语言常用的编程环境有 Visual C++ 6.0 和 Turbo C。用来编写程序的软件称为编译器。后来新兴起的编译器层出不穷，例如“啊哈 C”、“C-Free”等。除此之外，还有一些在线的编译器可以使用。对初学者来说，选择编译器要根据自己的使用习惯。编译器只是一个编写程序的工具，能善于使用、借助一个自己使用舒服的编译器来编写出出色的程序，才是终极目标。

3.4. 如何学习 C 语言

a. 锻炼分析解决问题的能力

学习程序设计，解决问题过程中更多的应该是对程序的思考。解决实际问题，就是对实际问题进行具体分析，建立相应的数学模型，确定一个实施方案。最后将实施方案转化成程序代码。如果具备了这种分析解决问题的能力，编写代码会变得非常简单。要提高这种能力，要在解决问题的过程中锻炼自己的“编程思想”。（相关资料可以查询《数学（必修 3）》（人民教育出版社）“程序框图”部分）

b. 多实践，提高动手能力

“纸上得来终觉浅”，既然程序设计是一门实践性的课程，所有的学习不能只在纸面上进行，应该多加实践，经常给自己找一些数学问题来做练习，提高动手能力。

c. 多阅读他人编写的程序

不要一味地埋头苦干，偶尔抬起头来看看别人写的代码，对比自己的程序，思考自己有哪些不足，取长补短，会使自己的代码越来越出色。

d. 养成良好的程序设计风格

请参阅《C 语言代码编写规范》，熟悉代码的编写格式。虽然代码规范与否并不影响程序的运行，但是依照代码的标准格式编写出的代码清晰明了，让人一看就能明白编写者的目的。通常一个团体、公司、机构会有一套自己独有的代码规范格式，用于内部编写代码使用。

二、 算法

1. 什么是算法

所谓算法，是解决一类问题的方法。做任何事情都要有一定的步骤，广义地说，为解决一个问题而采取的方法和步骤就称之为“算法”。对同一个问题可以有多种算法。

为了有效地解决问题，不仅要保证算法正确，还要考虑到算法的质量，选择合适的算法。算法通常分为数值运算算法与非数值运算算法，对于数值运算算法很简单，找到对应的数学模型，选择对应的算法，从而解决问题；对于非数值运算，种类繁多，要求各异，很难做到全部问题都有相应的答案。这个时候解题者需要参考已有的类似的算法，重新设计解决特定问题的专门算法。在解决这类问题的时候要学会举一反三，同时在脑中积累典型算法，搭建自己的“算法库”。

2. 算法的特性

2.1. 有穷性

一个算法应该包含有限的操作步骤，而不能让程序进入无休止的工作。

2.2. 确定性

算法中的每一个步骤都应当是确定的，也就是说，算法的含义应当是唯一的，而不应当产生歧义。

2.3. 有效性

算法中的每一个步骤都应当有效地执行，并得到确定的结果。例如当 b = 0，a / b 则不能有效执行。

2.4. 可以没输入，但要有输出

一个算法可以有零个或多个输入，一个或多个输出。输出很重要，是一个算法进行下来的结果。如果没有输出，算法就失去了意义。

3. 表示一个算法

3.1. 用自然语言

即用我们日常交流用的语言来表示解决问题的方法。由于这种方式通俗易懂，但文字冗长，意义不严格，往往在除了表示那些很简单的算法以外，通常不使用自然语言来表示一个算法。

3.2. 用流程图

相关内容可以参考《数学(必修 3)》(人民教育出版社） “程序框图”部分。

3.3. 用伪代码

伪代码书写格式自由，只是为了表达设计者的思想。例如求 5!：

```
1. begin
2. 1 -> t
3. 2 -> i
4. while  i ≤ 5
5.        {
6.          t*i -> t
7.          i+1 -> i
8.        }
9.      print   t
10. end
```

用伪代码书写算法形式上很宽松，容易书写，也容易看懂，但是没有办法放入编译器中生成真正的计算机程序，只能用来表示设计过程。因此，书写一个算法，最终还是要用到计算机语言。

3.4. 用计算机语言

例：将 3.3 的算法（求 5!）用 C 语言表示

```
1.  #include<stdio.h>
2.  int main( )
3.  {
4.        int   i , t;
5.        t = 1;
6.        i = 2;
7.        while(i <=5)
8.       {
9.              t = t * i;
10.             i = i + 1;
11.       }
12.       printf(“%d\n” , t);
13.       return   0;
14. }
```

第二部分：数据类型与表达式

一、 数据类型

1. 常量

程序运行中值不能被改变的量称为常量。数值常量就是数学中的常数。

1.1. 整型常量：整数即为整型常量。

1.2. 实型常量：实数即为实型常量。常量有两种表示形式：

第一种是十进制小数形式，由数字和小数点组成，如：123.456，0.345 等。第二种是指数形式，如 12.34e3 表示 12.34 x 10^3 。由于计算机输入时无法表示上下角标，所以规定字母 e 或 E 代表以 10 为底的指数。需要注意的是，e 或 E 之前必须有数字，e 后面必须为整数。

1.3. 字符常量

a. 普通字符

用单撇号括起来的单个的一个字符称为普通字符常量，如'a'，'Z '。但'ab'，'1a2' 不是字符常量。字符常量在计算机中存储时，不是以字符形式存储，而是以 ASCII 代码形式存储。例如'a'的代码是 97。ASCII 字符对照表见附录。

b. 转义字符

除了正常的字符之外，C 语言中还有一种特殊形式的字符常量，称为转义字符。转义字符是为了表示一些无法直接用字符表达的内容而存在的。例如在输出函数中，'\n'代表一个换行符。常用的转义字符如下表：

转义字符	字符值	输出结果
\'	一个单撇号（'）	输出具有此八进制码的字符
\"	一个双撇号（"）	输出此字符
\?	一个问号（?）	输出此字符
\\	一个反斜线（\）	输出此字符
\a	警告（alert）	声音或视觉信号
\b	退格（backspace）	后退一个字符
\f	换页（form deed）	将当前位置移至下一页开头
\n	换行	将当前位置移至下一页开头
\r	回车（carriage return）	将当前位置移至本行的开头
\t	水平制表符	将当前位置移至下一个 tab
\v	垂直制表符	将当前位置移至下一个垂直制表对齐点

表：常用的转义字符

c. 字符串常量

如"boy"，"123"等用双引号括在一起的多个字符称为字符串。

d. 符号常量与预定义指令

用#define 指令，在程序开始的时候讲程序中的一个符号替换成一个常量。例如：

```
1. #define      PI      3.14
```

使用预定义指令的好处在于，能在程序某个变量需要修改的时候做到改动一处就能修改全局。这个时候有一个需要思考的问题，例如有两行代码：

```
1. #define      PI      3.14
2. int    PI = 3.14;
```

当我们想要修改 PI 的值时，同样都可以做到“一改全改”，那么这两种方法有什么区别呢？区别是这样的，#define 预定义指令相当于我们直接在代码上进行“查找与替换”，属于直接在代码上进行修改，在预编译的时候这个符号就全部被替换成相应的值了，并不占用内存。但第二行代码中，如果修改了 PI 的值，就改动了对应内存空间的数值。

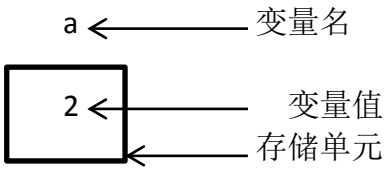
2. 变量

2.1. 变量与常变量

变量就是程序中定义的可以存入数据的存储空间，例如如下代码：

```
1. int    a;
2. a = 2;
```

就是在内存中开辟了一个名为 a 的内存空间，再为其存入相应的值。示意图如下：



图：存储单元的示意图

变量必须先定义，后使用。没有被定义的变量是不允许赋值的，相反地，没有值的变量是没有意义的。对程序编译连接的时候，编译系统会为每一个变量名分配对应的内存地址。从中取值，实际上是通过变量名找到相应的内存地址，再从中读取数据。

C 99 中允许使用常变量，例如：

```
1. const    int    a = 3;
```

这样就定义了一个整形变量 a，但是变量存在期间它的值不允许被改变。这样我们称 a 是一个常变量。

2.2. 标识符

在高级语言中，用来对变量、函数、类型等命名的字段统称为**标识符**（identifier），简单来说，标识符就是一段内存空间、一个变量、一个功能的名字。之前提到过的 printf、PI 等都是标识符。

C 语言规定，标识符只能由字母、数字、下划线构成，并且数字不能位于第一个字符。下面列出的都是合法标识符：

sun，average，_total，Student_1

下面列出的是不合法标识符：

M.J.Tom，3D86，$34

标识符在编译系统看来，同一个字母的大写与小写字母是两个不同的字母，Sum 与 sum 也是两个不同的变量名。按照人们的日常习惯，常常用小写字母来命名变量。

3. 基本类型

在之前的例程中，我们发现，定义一个变量的时候要指定相应的变量类型。C 语言中要求定义所有的变量都要指定相应的类型。这是为什么呢？

在数学运算中，数字是抽象存在的，并且允许模糊的计算存在，比如 10 / 3 = 3.333333……，但是在计算机运算中，数值是具体存在于某一个内存单元的。并且，储存单元的大小是有限的，这样的话在计算机中就不存在无穷的概念。例如有如下程序：

```
1.  printf("%d" , 1.0 / 3.0);
```

输出结果将会是 0.333333，只有 6 位小数，没有无限循环的小数，这就与“%d”（整形）所表示的变量类型的存储能力有关。所谓变量类型，就是计算机对变量分配的存储空间的长度与存储形式。

3.1. 整形 int

整形数据在内存中分配 2 个或 4 个字节，这与编译环境有关。Turbo C 2.0 为 int 型数据分配 2 个字节的存储空间，Visual C++为 int 型分配 4 个字节的存储空间。int 型能容纳的数值范围为-2^{31}~$(2^{31}-1)$（相应的存储方式的细节可以查阅参考资料[1]的 3.2.3 节（44 页）：《整形数据》）。

除了普通的 int 型数据之外，还有短整型（short int）和长整型（long int）数据类型，前者占 2 字节，后者占 4 个字节。由于实际编程需要，双长整型（long long int）新增进来。相信未来为了实际需要，还会加入更强大的变量类型。

常见的 int 型变量的储存空间和数值范围：

类型	字节数	数值范围
int 基本整形	2	-32678~32767
	4	-2147483648~2147483647
unsigned int 无符号基本整形	2	0~65535
	4	0~4294967295
short 短整型	2	-32768~32767
unsigned short 无符号短整型	2	0~65535
long 长整型	4	-2147483648~2147483647
unsigned long 无符号长整型	4	0~4294967295
long long 双长型	8	-9223372036854775808~9223372036854775807
unsigned long long 无符号双长整型	2	0~18446744073709551615

表：常见的 int 型变量的储存空间和数值范围

3.2. 浮点型 float

浮点型数据是用来存储具有小数点的实数的。由于同一个小数用指数形式表示的时候有多种形式，例如 3.14159 可以表示为 3.14159e0、0.34159e1，它们代表同一个值。可以看到小数点是在数字之间浮动的，所以实数的指数形式称为浮点数。

规定小数点左边的数字为 0，小数点右边的第一位数字不为 0，这种小数的表示形式称为规范化的指数形式，例如 0.314159e1。一个实数只有一个规范化的指数形式，在程序输出时以这种形式输出。

浮点数类型包括 float 单精度浮点型、double 双精度浮点型、long double 长精度浮点型。每种类型的存储方式各不相同，能表示的数值范围也各不相同，如下表：

类型	字节数	有效数字	数值范围（绝对值）
float	4	6	0 及 1.2e-38~3.4e38
double	8	15	0 及 2.3e-308~1.7e308
long double	8	15	0 及 2.3e-308~1.7e308
	16	19	0 及 3.4e-4932~1.1e4932

表：实型数据的有关情况

关于存储形式，可以参阅参考资料[1]的 3.2.5 节（50 页）:《浮点型数据》）

3.3. 字符型 char

字符与字符代码不是任意写一个字符程序都能识别，有些时候为了表示一些系统字符集中没有的字符，只能使用系统的字符集。如圆周率 π 。目前大多数系统使用 ASCII 字符集，所有字符集大多都包含全部的 127 个基本字符。（参见附录 1：ASCII 码对照表）

标准的字符型数据在系统中都用 7 位二进制数存放，例如数字 1 在 ASCII 码对照表中排在第 49 位，二进制码为 00110001。在 C 中，规定用 1 个字节（8 位）存储一个字符，字节中的第一位为 0。有关于字符型数据在计算机中的详细存储方式，可以参阅参考资料[1]的 3.2.4 节（48 页）:《字符型数据》）

字符变量是用类型标识符 char 定义的变量，比如定义一个字符型变量：

```
1. char c = '?';
```

'?'是字符型变量，它的 ASCII 编码为 63，所以在执行这个语句的时候，系统将整数 63 存入变量 c。此外，字符型数据的存储空间和数值范围如下：

类型	字节数	数值范围
signed char（有符号字符型）	1	-128~127
unsigned char（无符号字符型）	1	0~255

表：字符型数据的存储空间和数值范围

此外，关于字符型变量用整数形式存储的相关知识将在《输入与输出》一节详细讲解。

4. 其他类型

除了上面提到的基本类型之外，还有指针类型、函数类型、数组类型、结构体类型、共用体类型、枚举类型、空类型等。这些数据类型将在后面分别详细讲解。

二、 运算符

1. 算术运算符与算术表达式

几乎每一个程序都需要运算，否则程序就变得没有意义。要进行运算，就需要规定可以使用的运算符。C 语言的运算符包含除了控制语句和输入输出以外的基本操作，基本的算术运算符如下表：

运算符与举例	名称	目	运算结果
+a	正号运算符	1	a 的值
-a	负号运算符	1	a 的负值
a*b	乘法运算符	2	a 和 b 的乘积
a/b	除法运算符	2	a 除以 b 的商
a%b	求余（取模）运算符	2	a 除以 b 的余数
a+b	加法运算符	2	a 和 b 的和
a-b	减法运算符	2	a 和 b 的差
a++	自增运算符	1	使用 a 之后将 a 的值增加 1

++a			将 a 的值增加 1 再使用
a-- --a	自减运算符	1	使用 a 之后将 a 的值减小 1 将 a 的值减小 1 再使用
a?b:c	选择运算符	3	判断 a 的真假，如果真执行 b，假则执行 c

表：几种基本运算符

在这里需要一些相关说明：

1. 由于键盘无法输入乘除号，所以以 * 和 / 号代替。
2. 两个实数相除的结果是实数，两个整数相除是整数。但“取整”的舍去规则跟编译环境有关，有的编译器是向前取整，有的编译器是向后取整。例如-5 / 3 的结果在有的编译器里是-1，有的是-2。
3. 取模操作（%）要求参加运算的两个对象都是整数，取模运算结果也是整数，例如 5 % 3 = 2。除%之外的所有运算，参加运算的对象都可以是任何类型。
4. 自增（自减）运算符的运算与调用的先后顺序很重要，这一部分会在日后的《输入与输出》一节详细讲解。
5. 有关于不同类型间的混合运算，请参阅参考资料[1]的 3.2.7 节（54 页）。

在进行数值运算时，总会因为参加运算的两个数值类型不同而得不到正确的结果，这个时候可以对数据进行强制类型转换。强制类型转换符由一个括号加变量类型组成，例如：

```
1.  (double)a
```

即为将变量 a 转换成 double 类型。一般形式为：

```
1.  (类型名)(表达式);
```

除了算术运算符以外，C 语言还提供其他运算符，例如关系运算符、逻辑运算符、位运算符、逗号运算符、求字节数运算符等。相关知识读者可以自己查阅资料进行详细了解。

带有算术运算符组成的表达式叫做算术表达式，每一个算术表达式都有它的值。

2. 逻辑运算符与逻辑表达式

逻辑运算表达的是“关系”，逻辑表达式是一句可以判断真假的表达式。常用的逻辑运算符有三种：

运算符	含义	举例	说明
&&	逻辑与	a&&b	只有在 a 与 b 都为真时为真
||	逻辑或	a||b	a 和 b 之间有一个为真即为真
!	逻辑非	!a	!a 与 a 真值相反

表：常用的逻辑运算符

逻辑与和逻辑或为双目运算符，要求有两个对象参加运算。当参加运算的 a 和 b 为不同值的时候，逻辑运算得到的值也不相同：

a	b	!a	!b	a&&b	a||b
T	T	F	F	T	T
T	F	F	T	F	T
F	T	T	F	F	T
F	F	T	T	F	F

表：逻辑运算的真值表

另外，逻辑运算有相应的运算顺序，最低为赋值（=）运算，第二是&&和||，第三是关系运算符，第四是算术运算符，最高是非（!）运算。

三、 语句

一个 C 语言程序由若干语句构成，语句大概可分为控制语句、函数调用语句、表达式语句、空语句。所有语句都用分号作为一个或一块语句的结束标志。

1. 控制语句

控制语句在程序中完成一定的流程控制功能，相关内容将在第三部分：流程控制中详细讲解。C 语言中共有 9 种控制结构，分别完成条件、循环、选择、跳转等功能。

2. 函数调用语句

函数调用语句由函数名加括号加分号组成，例如我们在最开始见过的第一个程序：

```
1. printf("This is my NO.1 program");
```

3. 表达式语句

表达式语句由一个表达式和一个分号构成，这里需要明确表达式与语句的区别。例如：

```
1. a = 3
2. a = 3;
```

其中，第一行是表达式，第二行是语句。这里体现到了分号的作用，分号是一个语句中不可缺少的一部分。函数调用语句在某种意义上也属于表达式语句，例如：

```
1. sin(x)
```

也是表达式语句的一种，同时它也是函数调用语句。在学习 C 语言时不要拘泥与概念，灵活学习语言，注重在应用上才是正确的学习方法。

4. 空语句

下面是一句空语句：

```
1. ;
```

空语句仅由一个分号单独构成，它单独是一个语句，但它什么也不做。空语句常常用来做循环结构中的条件语句或循环体，表示这个循环体什么也不做。

5. 语句块

用花括号{ }将一些语句包含在一起就组成了语句块，也叫做复合语句。这种结构常常用于选择结构和循环结构。这时需要程序连续执行一组语句。例如：

```
1. {
2.     float  pi = 3.14159,r = 2.5 , area;
3.     area = pi * r * r;
4.     printf("area = %f" , area);
5. }
```

四、 输入与输出

输入与输出需要用到一个头文件 stdio.h，在程序开头要用预处理指令将这个头文件包含进去：

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

这个头文件中其中包含的两个库函数：printf 与 scanf，分别负责输出与输入。除此之外，C 语言中的标准输入输出函数还有其他几个函数，分别是：putchar（输出字符）、getchar（输入字符）、puts（输出字符串）、gets（输入字符串）。

首先要对输入输出做一些说明：

a. 所谓输入与输出是相对于计算机本身而言的，从计算机像输出设备（显示器、打印机等）输出数据称为输出，从输入设备（键盘、光盘、扫描仪等）像计算机输入数据称为输入。
b. C 语言本身不提供输出与输出函数，这两个函数是 C 语言标准库中提供的。
c. 在使用这两个函数时，一定要在程序开头用预处理指令#include<stdio.h>将这个头文件包含进去。

1. 标准输出函数 printf

标准输出函数的一般格式为：

```
1. printf(“格式控制”，输出表列);
```

格式控制是由双引号括起来的一个字符串，称为格式控制字符串，它包含两个部分，分别是格式声明（如%d，%f）和普通字符，普通字符即为在输出时照原样输出的字符。“输出表列”是程序要输出的变量名或表达式，输出表列格式声明符一一对应。

由于输出函数是函数，所以格式控制和输出表列都是这个函数的参数。这样的话，标准输出函数的一般形式可以表示为：

```
1. printf(参数 1，参数 2，参数 3，…… ，参数 n);
```

执行输出语句时，参数 2~参数 n 按照参数 1 中所指定的格式输出。参数 1 是必须要有的，参数 2~参数 n 是可选的。下面是一个标准输出函数的例子：

```
1. #include<stdio.h>
2. int  main( )
3. {
4.      int  a = 1 , b = 2 , c = 3 , d = 4 , e = 5 , f = 6 , g = 7;
5.      printf(“Number  is  %d%d%d%d%d%d%d\n” , a , b , c , d , e , f , g);
6.      return  0;
7. }
```

这个程序的执行结果为：

```
Number  is  1234567
```

需要提到的是，格式控制符决定输出表列中变量里存放的数据如何输出。前面说过，一切形式的数据在计算机中都是以二进制形式的编码进行存放，也就是说，在计算机的存储结构中，一切的数据的存储方式都是一样的。在输入时，通

过格式控制将不同格式的数据转换成二进制编码进行存储，在输出时，将二进制编码按照需要的格式进行输出。上述内容可以用以下例程说明：

```
1.  int  a = 64;
2.  printf(“%d、%c”,a,a);
```

上述输出语句输出结果是：

64、@

C 语言中的格式控制符大概有以下几种：

格式控制符	说明
%d、%i	以带符号十进制形式输出整数
%o	以八进制无符号形式输出
%x、%X	以十六进制无符号形式输出整数，X、x 分别代表输出时字母大小写
%u	以无符号十进制形式输出整数
%c	以字符形式输出
%s	以字符串形式输出
%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，E、e 代表输出时 E（e）的大小写

表：进行输出时的常用格式控制符

需要特别提到的是，用%f 输出时，可以控制输出宽度与保留的小数位数。例如%5.2f，表示输出时总共保留 5 位有效数字，2 位小数。例如：

```
1.  float  a=3.14159;
2.  printf(“%2.1f” , a);
```

这个输出语句的运行结果是：

3.1

可以在输出函数中参数 1 的位置使用转义字符。由于格式控制符由百分号加一个字母组成，如果想输出百分号的话，需要用两个百分号（%%）表示。例如：

```
1.  printf(“%f%%” , 1.0 / 3);
```

输出结果为

0.333333%

2. 标准输入函数 scanf

标准输入函数的一般形式为：

```
1. scanf(“格式控制”, 地址表列);
```

scanf 函数的第一个参数为格式控制，与输出函数不同的是，用户在用 scanf 输入的时候，必须严格按照 scanf 函数中格式控制字符串中的格式输入。scanf 函数中的格式控制符与 printf 函数中大致相同。

需要注意到的问题是，scanf 函数中参数 2~参数 n 是“地址表列”，而不是单单的变量名表列。例如：

```
1. scanf(“%d%d%d”, a, b, c);
```

这个语句是错误的，应该改为：

```
1. scanf(“%d%d%d”, &a, &b, &c);
```

“&”符号称为“取地址符”，“&a”代表变量 a 的地址。关于地址的相关内容请参阅第五部分：数组与指针。现在我们只需要知道应该为 scanf 函数提供你想为其输入值的地址即可。

3. 字符数据的输入和输出

输入输出字符数据时用到的函数是 putchar 和 getchar，这两个函数分别只能输出和接受一个字符。例如：

```
1. char  a,b,c;
2. a = getchar( );
3. b = getchar( );
4. c = getchar( );
5. putchar(a);
6. putchar(b);
7. putchar(c);
```

在 2~4 行分别为 a、b、c 输入一个字符，完成输入后会把 a、b、c 存放的字符原样输出。再例如：

```
1. int  a = 65, b = 66, c = 67;
2. putchar(a);
3. putchar(b);
4. putchar(c);
```

程序的输出结果是：

ABC

按照同样的方式，putchar 也可以输出转义字符。

五、 练习题

1. 设计一个程序，可以提示用户输入圆的半径，然后程序给出圆的面积

2. 设计一个程序，提示用户输入二次函数的系数 a、b、c，程序给出二次函数的解。已知二次函数求根公式：$x = \frac{-b \pm \sqrt{b^2-4ac}}{2a}$

3. 思考下面每一行语句执行后 a 的值与 printf 输出的值：

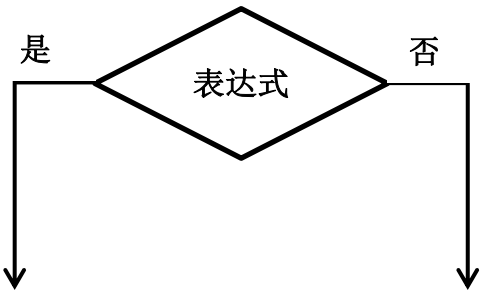
```
1. int  a = 7;
2. printf("%d" , a ++);
3. printf("%d" , a --);
4. printf("%d" , ++ a);
5. printf("%d" , -- a);
```

第三部分：流程控制

一、 选择结构

1. if…else 选择结构

在高中数学中，我们学过这样的程序结构：



图：选择结构流程图

这个流程图的程序运行过程是：当表达式为真，则沿着左侧的箭头向下执行，否则沿着右侧的箭头向下执行。例如：

```
1.  #include<stdio.h>
2.  int   main( )
3.  {
4.        int   a;
5.        printf("请输入你的性别，1 为男，2 为女：\n");
6.        scanf("%d",&a);
7.        if(a = 1)
8.        {
9.              printf("你是男生\n");
10.       }
11.       if(a = 2)
12.       {
13.             printf("你是女生\n");
14.       }
15.       else
16.       {
17.             printf("你确定没有输入错？\n");
18.       }
19.       return 0;
20. }
```

上述程序运行时，先提示用户输入数字，然后按照要求输出男或女。这是一个选择结构的典例。选择结构的一般形式如下：

```
1.  if(表达式 1)
2.        语句 1;
3.  if(表达式 2)
4.        语句 2;
```

```
5.  if(表达式 3)
6.      语句 3;
7.  …
8.  if(表达式 n)
9.      语句 n;
10. else
11.     语句 n+1;
```

需要注意的是，表达式 1~表达式 n 都是逻辑表达式，即能判断正误的表达式。当表达式的值为 1（TRUE）则执行此 if 下的语句，否则继续执行下一个 if 的判断。当语句 1~语句 n 为多条语句时，则要用花括号将其括在一起变成语句体。

2. switch 多分支选择结构

switch-case-default 结构可以实现多分支选择。例如要编写一个程序，根据学生的成绩等级来输出成绩区间：：

```
1.  int   main()
2.  {
3.     char   grade;
4.       printf("输入你的成绩等级：\n");
5.       scanf("%c",&grade);
6.       switch(grade)
7.       {
8.            case 'A' : printf("80~100\n");  break;
9.            case 'B' : printf("60~80\n");   break;
10.           case 'C' : printf("40~60\n");   break;
11.           case "D" : printf(0~40\n);      break;
12.           default : printf("输入错误");
13.      }
14.      return   0;
15. }
```

当程序遇到 switch 关键字时，会将括号中的变量与 switch 语句体中的每个 case 匹配，执行对应的语句。每个 case 语句后都有一个 break，它的作用是跳出这个 case 转到 switch 语句体结束。如果没有 break 的话，会在匹配到的 case 依次向下执行。

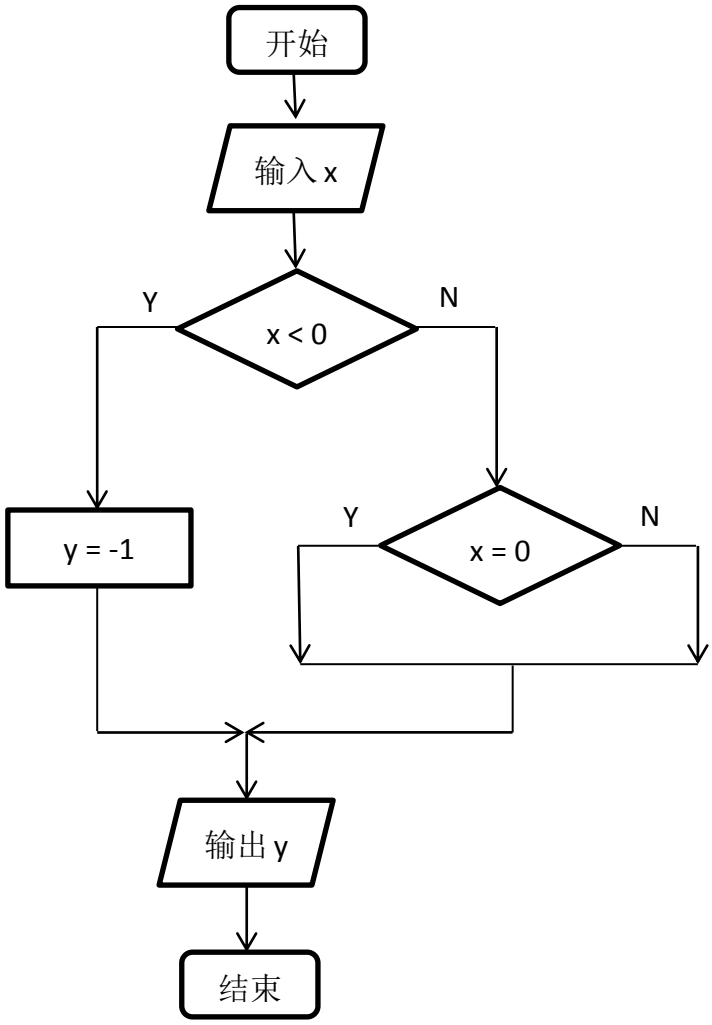
3. 选择结构的嵌套

if 和 switch 两种选择结构可以互相嵌套，形成嵌套结构。例如：

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

```
3.  {
4.      int   x , y;
5.      scanf("%d" , &x);
6.      if(x < 0)
7.          y = -1
8.      else
9.          if(x == 0)
10.             y = 0;
11.         else
12.             y = 1;
13.     printf("x = %d , y = %d\n" , x , y);
14.     return   0;
15. }
```

如上程序的流程如下图：



图：if 的选择结构嵌套示意图

这个程序体现了 if 语句的嵌套结构。当我们在设计程序算法，遇到分支的情况时，需要合理设计选择结构的嵌套。这将会在日后的编程中渐渐体会到。

4. 改变选择结构的状态 break

在 switch 选择结构中，如果在每一个分支后面没有特定的说明，默认会在此分支执行结束之后跳转到下一个分支。例如在第三部分、一、2 的 switch 例程中，如果输入 A，且每一个 case 语句结束的位置都没有 break 的话，输出结果是：

80~100
60~80
40~60
0~40
输入错误

进行算法设计时，要根据具体需要来使用 break 来改变程序运行状态，来使算法适合要解决的问题。

二、 循环结构

首先要讲到为什么要使用循环结构。例如当我们需要求 1~100 自然数之和，这个时候是不能用常规方法手工输入 1~100 个整数然后让计算机去进行计算的。如果那样的话，有 1000 或者更多的数字需要计算时怎么办？这个时候就需要用到循环结构。例如上面的问题有这样的解决方法：

```
1.  #include<stdio.h>
2.  int   main( )
3.  {
4.        int   i , s;
5.        s = 0;
6.        for(i = 1 ; i <= 100 ; i ++)
7.        {
8.              s = s + i;
9.        }
10.       printf(“%d” , s);
11.       return   0;
12. }
```

这个程序的运行结果是：

5050

常用的循环结构有：for、while、do…while 三种，三种循环结构的功能相同，但各有各的特点，所以它们各自能实现不同的功能。同时它们各自可以互相嵌套，还可以改变循环的状态，所以，善用循环结构往往能减少巨大的工作量，让电脑代替人脑完成一些人脑无法完成的工作。

1. while 与 do…while 循环结构

while 结构的一般形式是：

```
1. while(表达式)
2.     语句;
```

这个结构的运行过程是，首先判断表达式的真假，如果为真，则执行语句。语句可以为语句体，此时需要用花括号将语句体括起来。在 while 后面执行的语句（体）称为循环体。

do…while 结构的一般形式是：

```
1. do
2.     语句;
3. while(表达式)
```

这个程序的运行过程是，首先执行语句（体），再判断表达式真假，如果表达式为真，再进行循环。

2. for 循环结构

除了上述的两种循环结构外，C 语言还提供了另外一种更加灵活的 for 循环结构。它完全可以代替 while 和 do…while 语句。for 语句的一般形式是：

```
1. for(表达式 1；表达式 2；表达式 3)
2.     语句;
```

三个表达式的作用都不相同。表达式 1 设置初始条件，执行一次，可以没有、一个、或者多个；表达式 2 为判断是否执行循环的条件，当表达式 2 的值为真时，执行循环体；表达式 3 为循环的变化量，可以没有、一个或多个，执行一次。所以 for 语句的一般形式可以理解为：

```
1. for(赋初值 ； 循环条件 ； 增量)
2.     语句;
```

for 语句的一种极端形式是：

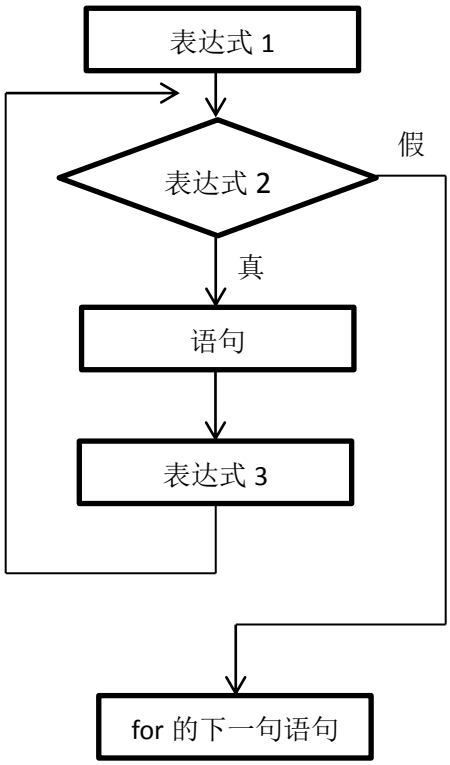
```
1. for( ; ; )
```

这句语句与

```
1. while(1)
```

是等价的，代表无条件循环。

for 循环体的执行过程是，先执行表达式 1，然后判断表达式 2 的真假，如果为真，执行语句，语句可以为语句体。语句（体）执行过后，执行表达式 3，for 的循环体到此执行完一次循环。整个过程如下图：



图：for 循环结构示意图

可以将 for 循环结构改写为 while 循环结构：

```
1. 表达式 1;
2. while (表达式 2)
3. {
4.     语句;
5.     表达式 3;
6. }
```

上述结构与 for 一般形式的循环结构完全等价。基于这种原理，我们可以将之前

提到的求 1~100 自然数和的程序改写：

```
1.  int   i , s = 0;
2.  while(i <= 100)
3.  {
4.        s = s + i;
5.        i ++;
6.  }
```

```
1.  int   i , s = 0;
2.  do
3.  {
4.        s = s + i;
5.        i ++;
6.  }
7.  while(i <= 100)
```

3. 循环结构的嵌套

上述三种循环结构的优缺点各不相同，在解决程序设计问题时各自能解决不同特征的问题。三种结构可以互相嵌套，实现更强大的功能。当进行设计比较复杂的循环问题时，应该注意代码的规范性，即每层嵌套之间有一个 Tab 键的缩进，这样的话代码看上去有层次感，一眼望去能看出都哪几行代码属于一个层次。便于别人阅读也便于自己修改。

具体的循环嵌套问题还要在日后解决问题中慢慢理解。

4. 改变循环结构的状态

当我们解决问题时，往往会遇到需要提前终止循环的情况。例如我们想要求 1~一个很大的自然数中所有质数的和，这样我们设置一个 1~很大的数的循环，每次遇到质数就把这个数加在和变量中，判断不是质数的话就跳过本次循环。这种方式叫做改变循环的状态。

在 C 语言中，有两种改变循环状态的方式：break 与 continue。为了讲解这两种方式，我们分别举不同的例子。

例如，我们在全校 1000 名同学范围内筹集资金，当资金数累计到十万元的时候就停止筹集。统计此时捐款人数。我们可以这样编写程序：

```
1.  #include<stdio.h>
2.  int   main( )
3.  {
4.        float   num , sum = 0;
5.        int   i;
6.        for(i = 0 ; i < 1000 ; i ++)
```

```
7.      {
8.          printf("输入捐款金额：");
9.          scanf("%f" , &num);
10.         printf("\n");
11.         sum = sum + num;
12.         if(sum >= 100000)
13.             break;
14.     }
15.     printf("有%d 人参加捐款。\n" , i);
16.     return   0;
17. }
```

在程序中可以看出，在第 13 行程序判断 sum >= 100000 是否成立，成立的话程序遇到 break，跳出整个循环。再看下面的例子。输出 1~100 之间可以被 3 整除的数，程序如下：

```
1.  #include<stdio.h>
2.  int   main( )
3.  {
4.      int   i;
5.      for(i = 0 ; i <= 100 ; i ++)
6.      {
7.          if(i % 3 != 0)
8.              continue;
9.          printf("%d\t" , i);
10.     }
11.     printf("输出完毕！\n");
12.     return   0;
13. }
```

在程序中可以看出，当程序遇到 continue 时，跳过本次循环，但循环还在继续。这时候需要自己体会二者的差别。

三、 练习题

1. 输出 1~1000 整数范围内所有的“水仙花数”。“水仙花数”即其各位数字的立方和等于其本身。例如 153 = 1^3 + 5^3 + 3^3 = 153，153 为一个水仙花数。
2. 求 1! + 2! + 3! + … + 20!的值。
3. 输出以下图形：

```
___*___
__***__
_*****_
*******
_*****_
__***__
___*___
```

（图中下划线为空格，这样写是为了看上去更加清晰。）

第四部分：函数与程序的模块化设计

一、 函数概述

利用之前学过的 C 语言语法，我们已经可以编写一些简单的程序了，但如果面对一些复杂的程序问题，如果将所有代码全都写在一个主函数里，会显得程序特别繁琐，没有可读性。例如我们想做一个简单的计算器，可以进行四则运算和三角函数、开方平方等运算，这个时候就需要有一种“组装”的思想，将每一个功能单独编写，然后在主函数中分别调用。这就涉及到 C 语言中“模块化程序设计”的设计思想。

在设计较大的程序时，往往将整个程序分为很多程序模块，分别写在对应的函数中，每个函数实现一个特定的功能。一个函数可以被其他函数调用多次。在一个程序中，只能有唯一的一个 main 函数。

我们举个例子来体会一下模块化程序设计的思想，例如我们想要写一个程序，可以输出在两个星号间的字符：

```
1.  #include<stdio.h>
2.  void   charwithstar(char   s)
3.  {
4.        printf(“**********\n%c\n**********” , s);
5.  }
6.  int   main( )
7.  {
8.        void   charwithstar(char   s);
9.        charwithstar(‘c’)
10.       return   0;
11. }
```

这个程序的运行结果是：

```
**********
    c
**********
```

上述程序中，charwithstar 是用户自己定义的一个函数，功能是输出一个夹在两行星号间的字符。在函数这一部分，有一些名词的概念需要明确。

参数：定义函数功能时用到的变量，用来指定函数的相应功能。例如 add 是我们定义的一个函数，它的功能是将 a 和 b 相加，那么 a 和 b 就是这个函数的参数。

返回值：函数的所有功能进行下来，需要保留到调用函数的函数中使用的值。比如刚才的 add 函数，我们需要将 a 和 b 的值加起来进行赋值，那么 a 和 b 的加和就是这个函数的返回值，就是整个函数的值。

C 程序从 main 函数开始，遇到其他函数时跳转到对应函数，执行完对应功能后回到 main 函数，最后在 main 函数中结束整个程序。函数与函数之间互相独立，不可以嵌套定义。从函数形式看，函数分为有参函数与无参函数。无参函数一般用来执行固定的操作，有参函数一般需要进行参数传递，对不同的参数进行处理输出。函数有的带回返回值，有的不带回返回值，这个因程序而异。

二、 函数的定义

定义一个函数即告诉编译系统你想要定义的函数的名字、功能、返回值类型、参数名字与类型，这样编译系统才能定义一个你期望功能的函数。通常定义一个函数的一般形式为：

```
1. 返回值类型名  函数名(参数类型名  参数名 , 更多参数……)
2. {
3.       函数体
4.       ……
5.       指定返回值
6. }
```

其中，如果没有参数，可以不写参数或者在参数的位置写 void，如果没有返回值，可以在返回值类型名的位置写 void。我们甚至可以用以下的方法来定义一个空函数：

```
1. void  empty( )
2. {     }
```

这是一个空函数，它什么也不做。在程序的实际设计中，往往需要在设计程序的最开始写好程序的大纲，这个时候就需要把每一个能想到的功能用空函数的形式写在指定的位置，这样做的话程序结构清晰，对日后的修改有益。

特别地，我们指定 main 函数的返回值是 int 型，在 main 函数末尾要让主函数返回 0 作为函数执行完毕的标志。有的参考资料中规定 main 函数是 void 型，此时 main 函数末尾不需要返回 0。这个因教材而异，在这里我们统一规定为前者。

三、 函数的声明与调用

定义一个函数当然是为了在其他位置使用这个函数。调用函数之前，必须要事先声明。这里的声明包括函数事先在预处理行中包含进去的情况，此外，需要自行定义。例如我们在程序开端定义了一个函数：

```
1. int  add(int  a , int  b)
2. {
3.      return  a + b;
4. }
```

它的功能是将 a 和 b 两个数相加。然而我们在 main 函数中调用这个函数的时候需要再声明一次：

```
1. int  main( )
2. {
3.      int  add(int  a , int  b);
4.      printf("%d" , add(1 , 2));
5.      return  0;
6. }
```

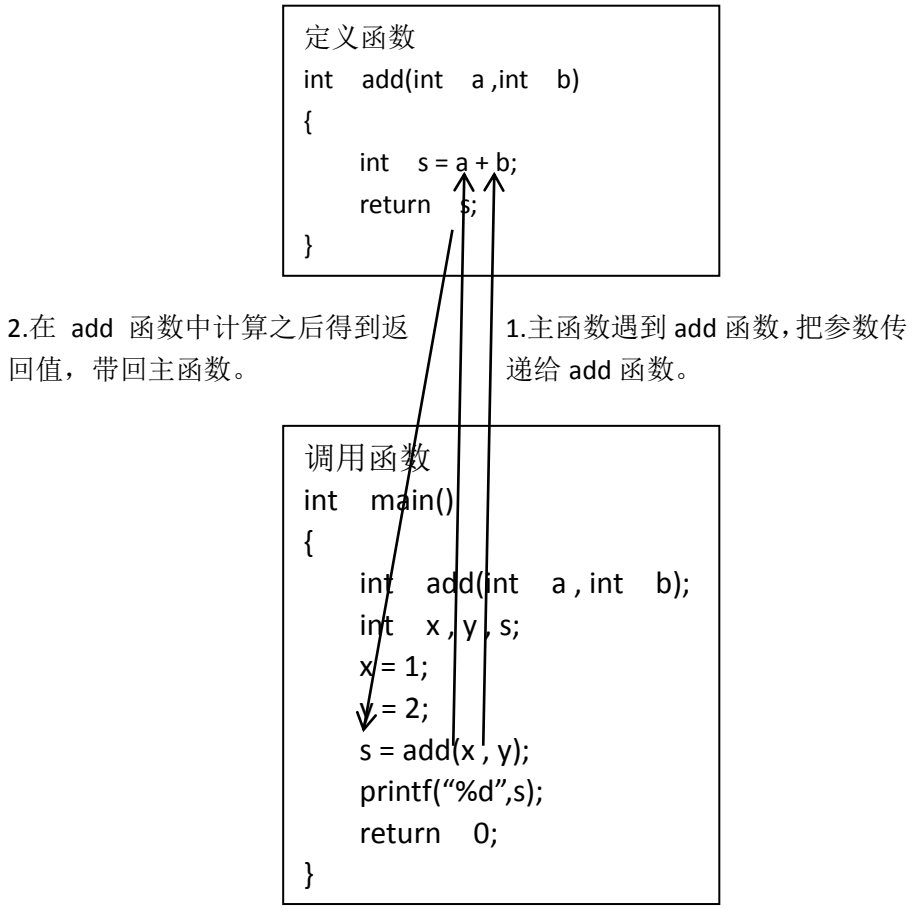
可以看到，在第 3 行类似将函数原型，只是多了一个分号，这就是函数的声明。函数在调用之前必须要进行声明。

函数调用的一般形式为：

```
1. 函数名(实参表列)
```

如果函数没有参数，可以省略实参表列，但括号必须有。函数有三种调用方式，其一是单独调用，一般用来实现一些操作；其二是作为函数表达式，函数的返回值作为一个值来使用；其三是作其他函数的参数。其二和其三实际上属于同一种。

函数在调用的时候，涉及“参数传递”的问题。当定义函数时，函数的参数称为“形式参数”（简称“形参”）。当调用函数时，被调用函数的参数称为“实际参数”，（简称“实参”）。调用函数时，实参传递给形参，这个过程叫参数的传递（简称“传参”）。具体过程如下图。



图：函数传递参数示意图

函数在调用的时候可以互相嵌套，就是将一个函数的返回值作为下一个函数的参数。例如 add 函数是两个数相加，multi 是两个数相乘。那么

1. add(multi(a , b) , multi(c , d));

则表示数学算式：

a * b + c * d

这就是函数的嵌套使用。

四、 函数的递归调用

在一个函数中直接或间接调用其函数本身叫做函数的递归调用，这是 C 语言的一大特色。例如下面这个求 n！的数学函数：

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

当我们求一个数的阶乘时，可以将问题转化为 n * (n - 1)!的问题，这样的话这个问题就转换成了一个经典的递归问题。例如我们要求 fac(3)：

fac(3) = 3 * fac(2)
fac(2) = 2 * fac(1)
fac(1) = 1

fac(2) = 2 * 1 = 2
fac(3) = 2 * 3 = 6

这样就完成了整个递归算法。这里我们可以发现，所有递推式都可以化为递归。一个递归问题分为两个部分，分别是递推与回归。由已知的结果递推，推到有已知值的式子之后再向原方向回归，最后得知结果。又例如下面这个求斐波那契数列第 n 项的函数：

```
1. int  fib(int  n)
2. {
3.      if(n == 0)  return  0;
4.      if(n == 1)  return  1;
5.      if(n >1)    return  fib(n - 1) + fib(n - 2);
6. }
```

在求解过程中，求 fib(n)的时候就要推到求解 fib(n - 1)和 fib(n - 2)的问题上，这样就又必须计算 fib(n - 3)和 fib(n - 4)，以此类推，最后将会推到 fib(1)和 fib(0)，能立即得到结果。这时候递推终止，再一路返回到 fib(n)的值。这样看来，递归一定要有结束条件，否则就是一串无止尽的循环。

递归运算的优点是源程序简洁，缺点也很显著，会消耗过多的内存与运行时间，效率不高。

五、 局部变量与全局变量

我们最常见的定义的变量都是在 main 函数中定义的，它们在整个 main 函数中都有效。也有的变量定义在我们自己定义的函数里，有没有想过，在其他的函数里可以使用这个变量吗？答案是不可以的。这就是变量作用域的问题。

定义变量时有三种情况：在函数外部定义、在函数开头定义、在函数中定义。在函数中复合语句内定义的变量只能在复合语句中使用能够，在函数开头定义的变量可以在整个函数内使用，这两种变量叫做“局部变量”。在函数外部定义的变量可以在整个程序中使用，这种变量叫做“全局变量”。例如有以下示意程序：

```
1.  int   m , n;
2.  func_1(int   a)
3.  {
4.      int   b , c;
5.      ……
6.      {
7.          int   i , j;
8.          ……
9.      }
10.     ……
11. }
```

在上述程序全部定义的变量中，m、n 是全局变量，可以在整个程序中使用。b、c 是函数 func_1 的局部变量，只能在这个函数中使用。i、j 是复合语句（5~8 行）中的局部变量，只能在这个语句块（一个花括号内）中使用。

六、 变量的声明与定义

因为变量的作用域不同，而我们有时候需要调用一些本来不在此作用域中的变量，就需要对已经定义好的变量进行声明。例如在如下的示意程序中：

```
1.  int   m , n;
2.  func_1(int   a)
3.  {
4.      int   b , c;
5.      {
6.          int   i , j;
7.          ……
8.      }
9.      ……
10. }
11. func_2(int   d)
12. {
13.     int   e;
14.     ……
15.     extern   b;
16. }
```

在第 15 行我们用 extern 关键字将变量 b 声明，这样我们就可以在 func_2 函数中

使用变量 b。使用 extern 关键字声明变量意味着我们将此变量的作用域扩展到此。

第五部分：数组

用我们之前学到的 C 语言语法，如果想要计算 30 个学生的平均成绩，我们需要定义 30 个 float 型变量，然后将它们加起来除以 30 就可以了。30 个变量的问题似乎不大，只需要定义 30 个变量（s1、s2、s3……s30）即可。换作是 1000 个变量怎么办呢？人们想了一个办法，就是用同一个名字代表具有相同属性的一群数据，再辅以下标来在它们彼此间区分他们。例如，S_1、S_2、S_3 代表学生 1、学生 2、学生 3。一群具有同属性的数据在一起称为“数组”。数组是一组有序数据的集合。

数组中各数据的排列是有一定规律的，下标代表数据在数组中的序号。用一个数组名和一个下标的组合（如 a2）来确定数组中唯一的元素，同样可以这样赋予数组实际意义。例如：S12 就可以代表第 12 位同学的成绩。数组中的每一个元素都属于同一个数据类型，不能把不是同一个类型的数据放在同一个数组中。

一、 一维数组

1. 一维数组的定义与引用

由于在编写程序的时候无法输入下表，所以在 C 语言中用数字加方括号表示下标。如下即是对一个一维数组的定义：

```
1. int  a[10];
```

它表示定义了一个一维数组，数组名为 a，这个数组有 10 个整形元素。定义一个一维数组的一般形式为：

```
1. 类型标识符  数组名[常量表达式]
```

需要注意的是，按照 int a[10]定义后这个数组中有十个元素，从 a[0]开始，到 a[9]结束，并没有 a[10]这个元素。“常量表达式”可以为常量与符号常量，如 a[3 + 5]，但不可以是 a[n]，也就是说，不允许对数组做动态定义。例如：

```
1.  int  x;
2.  scanf("%d",&x);
3.  int  a[x];
```

这种定义在 C 语言中是不允许的。

经过 int a[10]的定义，内存中为数组 a 划分出一片存储空间，这片存储空间

是连续的，可以存放 10 个整形元素。定义 10 个元素的数组的时候相当于定义了 10 个整形变量，显然比章首提到的方法方便。

当我们想要使用数组中的元素时，可以用：

1. 数组名[下标]

的形式来引用数组中的任意一个元素。例如我们想要对 10 个元素依次进行赋值，然后按倒序输出：

```
1.  #include<stdio.h>
2.  int   main( )
3.  {
4.        int   i , a[10];
5.        for(i = 0 ; i < 10 ; i ++)
6.              a[i] = i ;
7.        for(i = 9 ; i >= 0 ; i --)
8.              printf("%d" , a[i]);
9.        printf("\n");
10.       return   0;
11. }
```

2. 一维数组的初始化

在定义变量的同时，对数组元素赋值，称为数组的初始化。对一维数组初始化的方法有以下几种：

a. 定义数组时给全部元素赋值

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

此外，由于给全部元素赋值时长度已经确定，可以不指定数组长度。上面的代码等价于下面这句代码：

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

使用这种方法定义数组时，需要格外注意的是数组名后面的一对方括号不能省略，方括号是数组的标识符。

b. 定义数组时给部分元素赋值

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

这样的话，数组 a 前五个元素这样赋值，其余元素初值为 0。

c. 使数组元素全部为 0

用下面的方法可以将数组元素全部赋初值为 0：

```
1. int  a[10] = {0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0};
```

或者是：

```
1. int  a[10] = {0};
```

定义数组时应当指定数组长度并进行初始化，如果没有进行初始化，系统会自动将其初始化为 0。如果是字符型数组则初始化为'\0'，如果是指针数组则初始化为 NULL。

3. 应用举例

将数组中的元素按顺序重新存放。

在这里介绍“起泡排序法”，这种方法的基本思路是，将相邻的两个数比较，将较小的数放在前面。例如有 6 个数字：9、8、5、4、2、0，要求按从小到大的顺序排好。最开始数字是这样排序的：

9 8 5 4 2 0

我们第一趟将最顶端的数字 9 依次与后面的数字比较，比 9 大的放在右面，比 9 小的就与 9 交换位置。第一次~第六次的比较过程如下：（下划线表示交换位置）

9 8 5 4 2 0
8 9 5 4 2 0
8 5 9 4 2 0
8 5 4 9 2 0
8 5 4 2 9 0
8 5 4 2 0 9

可以看到，经过 6 次比较，原本在最顶端的数字 9 已经沉底，第二趟我们进行如下过程的比较：

8 5 4 2 0 9
5 8 4 2 0 9
5 4 8 2 0 9
5 4 2 8 0 9
5 4 2 0 8 9

第二趟中，原本沉底的数字 9 是不动的，在 9 沉底后在顶端的数字 8 经过第二趟比较，也沉在了仅次于 9 的低端。以此来推，经过 5 趟比较，就可以将原本的 6

个数字按照升序排好。可以分析出来，当有 n 个数时，需要进行 n – 1 趟比较。上述算法我们称作“气泡排序法”，算法的程序如下：（假设有 10 个数据）

```
1.  #include<stdio.h>
2.  int   main()
3.  {
4.        int   a[10];
5.        int   i , j , t;
6.        printf(“Input   10   numbers:\n”);
7.        for(i = 0 ; i < 10 ; i ++)
8.              scanf(“%d” , &a[i]);
9.        printf(“\n”);
10.       //起泡排序开始
11.       for(j = 0 ; j < 9 ; j ++)
12.             for(i = 0 ; i < 9 – j ; i ++)
13.             {
14.                   t = a[i];
15.                   a[i] = a[i + 1];
16.                   a[i + 1] = t;
17.             }
18.       //起泡排序结束
19.       printf(“The sorted numbers are:\n”);
20.       for(i = 0 ; i < 10 ; i ++)
21.             printf(“%d\t” , a[i]);
22.       printf(“\n”);
23.       return   0;
24. }
```

二、 二维数组与引申

当我们有一个表格，记录运动员各项比赛得分情况，横排是运动员姓名，竖排是项目名称，如下：

	高老大	王二	张三	李四	赵五	钱六
100 米跑	95	75	89	75	98	87
铅球	85	91	88	92	98	70
标枪	75	85	95	98	76	82

我们可以将这个表格的数据存放在二维数组中，例如我们可以用 S[1][2]代表

王二的铅球成绩，即 S[1][2] = 91。二维数组也称为矩阵，这样的数组具有两个维度：行和列。我们可以用类似定义一维数组的方式来定义一个二维数组：

```
1. int  a[3][3]
```

这样我们就定义了一个 3 x 3 的二维数组，写成矩阵形式如下：

a[0][0]	a[0][1]	a[0][2]
a[1][0]	a[1][1]	a[1][2]
a[2][0]	a[2][1]	a[2][2]

仍然需要注意的是，二维数组的两个维度标号都是从 0 开始的，这一点与一维数组相同。需要明确的概念是，二维数组写成矩阵形式是为了理解方便，但是在内存中，二维数组仍然是线性存储的。

对二维数组进行初始化时与一维数组有所不同，例如我们对上述的 3 x 3 数组进行初始化：

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

也可以将所有元素写在一个花括号内：

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

同样我们也可以对部分元素赋初值，这个时候其余元素为 0：

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

赋值结果是：

```
1  0  0
2  3  0
4  0  0
```

对二维数组进行赋初值时，要记得一个准则，那就是在一个花括号内的是同一行，隔一个花括号就是下一行。

以上就是二维数组的定义与调用方式，下面举一个例子来讲解二维数组的综合运用。例如我们需要将一个二维数组行列互换，存在另一个二维数组中。首先我们分析问题，这个问题的中心就在于将二维数组 a 的元素 a[i][j]存放在另外一个二维数组 b[j][i]中。假设有二维数组 a[2][3]，写程序如下：

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

```
3.  {
4.      int   i , j;
5.      int   a[2][3] = {0};
6.      for(i = 0 ; i <= 1 ; i ++)
7.          for(j = 0 ; j <=2 ; j ++)
8.              scanf("%d" , &a[i][j]);
9.      int   b[3][2] = {0};
10.     printf("a:\n");
11.     for(i = 0 ; i <= 1 ; i ++)
12.     {
13.         for(j = 0 ; j <=2 ; j ++)
14.         {
15.             printf("%d",a[i][j]);
16.             b[j][i] = a[i][j];
17.         }
18.         printf("\n");
19.     }
20.     printf("b:\n");
21.     for(i = 0 ; i <= 2 ; i ++)
22.     {
23.         for(j = 0 ; j <= 1 ; j ++)
24.             printf("%d",b[i][j]);
25.         printf("\n");
26.     }
27.     return   0;
28. }
```

二维数组的定义与调用方法可以引申为多维数组，只要有需要，数组可以有三维、四维……

三、 字符数组

C 语言中不存在字符串类型，字符串是存放在字符数组中。用来存放字符的数组是字符数组，我们可以像定义一维数组那样来定义一个字符数组：

```
1.  char   a[10];
```

这样我们定义了一个可以存放十个字符的数组 a。由于字符是以整数形式存储，因此我们也可以用整形的数组存放字符，例如：

```
1. int  a[10];
2. a[0] = 'a';
```

这样是合法的，但浪费存储空间。因为一个 int 型的存储空间是 4 字节，但一个字符只占用 1 字节，剩下的 3 字节是浪费的。

对字符数组进行初始化的方法也有很多中，其中最容易理解的方法是：

```
1. char  a[10] = {'W' , 'E' , 'L' , 'C' , 'O' , 'M' , 'E'};
```

这个时候，数组 a 的存放状态如下：

W	E	L	C	O	M	E	\0	\0	\0

需要提到的是，在字符数组中，最后一位一定以空字符（'\0'）结束，作为字符数组结束的标志。除此之外，我们还有另外一种初始化字符数组的方法：

```
1. char  a[] = "WELCOME";
```

用这种方法定义字符数组，是直接把字符串赋值给数组，字符串用双撇号括在一起。当用 printf 输出字符数组时，检查到第一个'\0'时即停止输出。当我们对字符数组进行输入输出时，可以对数组的某个元素进行输入输出，也可以对整个数组进行输出。例如：

```
1. char  a[] = "BOY";
2. printf("%s" , a);      //%s 意为以字符串形式输出/输入
3. printf("%c" , a[0]);    //用%c 输出/输入单个元素
```

此外，C 语言还提供了一些字符串处理函数：

函数名与调用方式	作用
puts(字符数组);	将一个字符串输出到屏幕上
gets(字符数组);	从输入终端获取一个字符串
strcat(字符数组 1，字符数组 2);	将字符串 2 接在字符串 1 的后面
strcpy(字符数组 1，字符串 2);	将字符串 2 赋值到字符串 1 里
strcmp(字符串 1，字符串 2);	字符串比较大小
strlen(字符串);	返回字符串的长度
srtlwr(字符串); strupr(字符串);	将字符串转换为大写/小写

当使用这些函数时，要在预处理部分将 string.h 头文件包含在程序中。

四、 练习题

函数部分的习题并在这里。

1. 写一个函数可以将两个字符串连接在一起，不用 stract 函数。
2. 求一个 3 x 3 整形矩阵对角线数字之和。
3. 有一篇英文文章，共 3 行，每行 80 个字符。要求统计出其中英文大写字母、小写字母、数字、空格以及其他字符的个数。
4. 写一个函数，可以给 3 x 3 的数组转置。
5. 写一个函数，可以将一个字符串倒序存放。

第六部分：指针初步

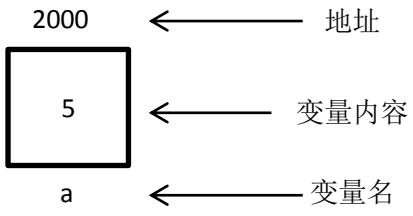
一、 指针概述

要讲述指针，首先要讲到数据在内存中的存储方式。在 C 编译系统中，系统为整形数据分配 4 字节内存空间，为字符型数据分配 1 字节内存空间。而内存区每个字节都有一个编号，这个编号就是这个字节的地址。由于通过地址可以找到存放这个变量的位置，所以说这个地址指向这个变量，通常我们称这个地址是变量的指针。

例如我们有如下代码：

```
1.  int   a = 5;
```

变量 a 在内存中的存放情况示意图如下：



其中，这个变量占用 4 个字节的内存空间，如果有下一个变量被声明，地址号则是从 2004 开始。地址不一定是从 2000 开始，这里只是示意。如果我们声明了一个整型数组，则可以更好地理解整形数据占用 4 个字节的问题：

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

地址	变量名	变量内容
2000	a[0]	1
2004	a[1]	2
2008	a[2]	3
2012	a[3]	4

当我们用 scanf 输入数据的时候需要用到’&’取地址符，例如：

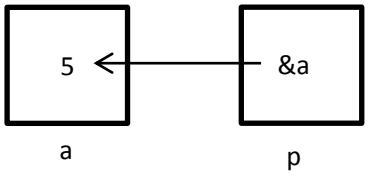
```
1. int  a;
2. scanf(“%d” , &a);
```

这个时候，系统将键盘输入的值送入 a 的地址指向的内存单元。如果有语句通过变量名直接调用这个变量，这叫做变量的“直接访问”。如果通过地址来访问变量的内容，这就叫做“间接访问”，这个时候就需要用到指针变量。

我们在 C 语言中定义过很多类型的变量，例如整形、浮点型、字符型，还有一种特殊的变量类型可以用来存放其他变量的地址，叫做指针变量，标识符是一个星号（*）。例如有指针变量 p，我们通过如下语句将变量 a 的地址存放给 p：

```
1. p = &a;
```

这个时候变量 a 与 p 在内存中的关系如下图：



可以看到，p 本身是一个变量，它存放的内容是 a 的地址，p 的内容指向变量 a。这个时候称 p 是指向 a 的指针变量。这样看来，指针变量就是可以存放其他变量地址的变量。

二、 定义与引用指针

定义指针变量的一般形式为：

```
1. 类型名  *指针变量名;
```

例如：

```
1. int   *p;
```

通过下面这个例子可以更加详细理解指针变量：

```
1.  #include<stdio.h>
2.  int   main( )
3.  {
4.        int   a = 2 , b = 3;
5.        int   *p1 , *p2;
6.        p1 = &a;
7.        p2 = &b;
8.        printf("a = %d , b = %d\n" , a , b);
9.        printf("*p1 = %d , *p2 = %d\n" , *p1 , *p2);
10.       return   0;
11. }
```

运行结果是：

```
a = 2 , b = 3
*p1 = 2 , *p2 = 3
```

定义指针后，要告诉系统这个指针指向的变量是谁，例如上例中的 6、7 行。给指针赋值后就可以通过指针来访问指针指向的变量了，在第 9 行，通过 * 号和指针变量名来代表这个指针变量指向的内容。

指针变量同时可以作为函数的参数，例如下面的例程，通过函数交换两个变量的值：

```
1.  #include<stdio.h>
2.  void   swap(int   *p1 , int   *p2)
3.  {
4.        int   temp;
5.        temp = *p1;
6.        *p1 = *p2;
7.        *p2 = temp;
8.  }
9.  int   main()
10. {
11.       void   swap(int   *p1 , int   *p2);
12.       int   *point1 , *point2;
13.       int   a = 1 , b = 2;
```

```
14.     point1 = &a;
15.     point2 = &b;
16.     printf("a = %d , b = %d\n" , *point1 , *point2);
17.     swap(point1 , point2);
18.     printf("a = %d , b = %d\n" , *point1 , *point2);
19.     return  0;
20. }
```

可以思考一个问题，main 函数中两个指针变量的指向改变了没有？

三、 指针与数组

我们已经知道，每个变量都有它自己的地址，由于数组是多个同类型的变量，所以每个数组元素都有它自己的地址，且在一个数组内的所有元素地址是连续的。数组元素的指针就是数组元素的地址。当我们想引用数组元素，可以用下标引用，这是数组元素的直接引用。通过指针引用数组是数组元素的间接引用。

特别地，C 语言中数组名代表数组首元素的地址。例如下面第 3、4 两行代码等价：

```
1.  int  *p;
2.  int  a[10];
3.  p = a;
4.  p = &a[0];
```

当指针指向数组元素的时候，对指针的运算概念与我们之前提到的四则运算有所不同。指针加一个整数 n 时，代表指针向前移动 n 个元素，减则为向后移动。同样也可以进行自加和自减运算。当两个指针都指向同一个数组中的元素时，两个指针相减则表示两个指针相隔的距离。例如：

```
1.  #include<stdio.h>
2.  int  main( )
3.  {
4.      int  *p;
5.      int  a[10];
6.      int  i;
7.      p = a;      //这行可以等价为  p = &a[0]  或*p = a[0]
8.      for(i = 0 ; i < 10 ; i ++)
9.          a[i] = i;
10.     for(i = 0 ; i < 10 ; i ++ , p ++)
11.         printf("%d" , *p);
12.     return  0;
```

```
13. }
```

特别地，当指针指向数组元素进行加减时，指针加一不是将 p 的值（地址）简单地加一，而是加上一个指针类型所占用内存大小的字节数。例如我们定义指针和数组如下：

```
1. char  a[] = “WELCOME”;
2. int  *p;
3. p = a;
```

注意到数组类型和指针类型不同，但我们将指针指向数组首元素。这个时候如果：

```
1. printf(“%c” , *p);
2. p ++;
3. printf(“%c” , *p);
```

第 1 行会输出字符数组的首字符 ’W’，但经过第 2 行指针移动后，第 3 行不会输出第二个字符而会输出第 5 个字符 ‘O’。这就是因为整形元素占用 4 个字节，而字符型元素占用 1 个字节。移动指针时按照指针类型移动了 4 个字节，相当于字符数组的 4 个元素的大小。

同时，指针还可以与之前学过的函数结合，例如数组首元素可以当成函数的参数，可以声明指针类型的数组，感兴趣的可以自己查阅参考文献 1 的教材。

四、 练习题

1. 结合之前提到的例程：

```
1. #include<stdio.h>
2. void  swap(int  *p1 , int  *p2)
3. {
4.       int  temp;
5.       temp = *p1;
6.       *p1 = *p2;
7.       *p2 = temp;
8. }
9. int  main()
10. {
11.      void  swap(int  *p1 , int  *p2);
12.      int  *point1 , *point2;
13.      int  a = 1 , b = 2;
```

```
14.     point1 = &a;
15.     point2 = &b;
16.     printf("a = %d , b = %d\n" , *point1 , *point2);
17.     swap(point1 , point2);
18.     printf("a = %d , b = %d\n" , *point1 , *point2);
19.     return  0;
20. }
```

思考，如果将 swap 函数改成如下：

```
1.  swap(int  x1 , int  x2)
2.  {
3.      int  temp;
4.      temp = x1;
5.      x1 = x2;
6.      x2 = temp;
7.  }
```

这个时候为什么不能完成值的交换？

本部分更多的练习将结合下一部分出现。

第七部分：自定义数据类型

一、 结构体的概念与定义

在前面的学习中，变量大多都是互相独立的，即使是内存中的地址也是互相毫无干系的。即使是数组，也只是要求其中的元素必须是同样的类型。那么可以不可以有一种结构，可以将一些不同类型的变量放在一起，而他们又具有一定的联系呢？例如一个学生的资料，其中有学号、姓名、性别、年龄、地址。C 语言中允许用户自己建立由不同类型数据组成的组合型数据结构，叫做“结构体”。可以理解为结构体也是一种数据类型，例如下面是一个自行创建的结构体类型：

```
1.  struct  Student
2.  {
3.      int  num;
4.      char  name[20];
5.      char  sex;
6.      int  age;
7.      char  add[30];
```

```
8. };
```

struct 是结构体类型的标识符，Student 是用户自定义的结构体名称，就像 int a 中的 a 是变量名称一样。这个结构体中有 5 个成员，分别是 num、name、sex、age、add，它们各自有自己的数据类型。这个结构体在内存中占 4 + 20 + 1 + 4 + 30 = 59 字节。上面定义的结构体可以这样示意：

Student				
num	name	sex	age	add

定义结构体的一般形式为：

```
1. struct  结构体名
2. {
3.     //成员表列
4.     类型名  成员名;
5. };
```

定义结构体的时候不要忘了末尾有分号。结构体表现的是“包含”关系，如上例即表示 Student 中包含了 num、name、sex、age、add 五个元素。当然，结构体的成员可以属于另外一个结构体，例如：

```
1. struct  Date
2. {
3.     int  month;
4.     int  day;
5.     int  year;
6. };
7. struct  Student
8. {
9.     char  name[20];
10.    int  age;
11.    char  sex;
12.    char  add[30];
13.    struct  Date  birthday;
14. };
```

这个时候，此结构体的结构如下图所示：

Student						
name	age	sex	add	birthday		
				month	day	year

二、 结构体类型的变量

当定义好了一个结构体，可以视为定义好了一个类型。比如学生是一个类型，这个类型具有学号姓名等属性。当我们定义好了这一类型的属性后，就要说明谁属于这个类型。比如王小明和张小红都是学生，这样它们就都具有学生这一类型应有的特征。这时候王小明和张小红就是学生这个结构体类型的变量。写在程序中如下：

```
1. struct  Student
2. {
3.       char  name[20];
4.       int  num;
5.       char  add[30];
6. }  hong , ming ;
```

这个时候 hong 和 ming 的结构如下图所示：

	name	num	add
hong			
ming			

声明结构体类型的变量有两种方法，其中一种是在定义结构体的时候直接声明结构体变量，一般形式如下：

```
1. struct  结构体名
2. {
3.       //成员表列
4.       类型名  成员名;
5. }  变量 1 , 变量 2 , …… , 变量 n ;
```

注意分号是在变量表列之后的。还有一种定义方式，就是在已经定义好结构体之后，直接使用结构体名当类型名来定义，如下：

```
1. struct  Student  ming;
```

声明结构体变量的同时也可以直接对其进行初始化，下面这个例子介绍了结构体变量初始化和引用的方法：

```
1.  #include<stdio.h>
2.  int   main( )
3.  {
4.        struct   Student
5.        {
6.              int   num;
7.              char   sex;
8.              char   name[20];
9.              char   add[30];
10.       } a = {1405010314 , 'M' , ming , "Beijing"};
11.       printf("number : %d\n" , a.num);
12.       printf("name : %s\n" , a.name);
13.       printf("sex : %c\n" , a.sex);
14.       printf("address : %s\n" , a.add);
15.       return   0;
16. }
```

程序的运行结果是：

```
number : 1405010314
name : ming
sex : M
address : Beijing
```

三、 结构体数组

结构体数组的概念很好理解，即一次性声明具有相同类型的结构体变量。例如：

```
1.  struct   Student
2.  {
3.        int   num;
4.        char   name[20];
5.  } stu[3] = {001 , "ming" , 002 , "hong" , 003 , "hua"} ;
```

当结构体数组成员很多的时候，可以不在声明的时候进行初始化。

四、 结构体与指针的结合

如果声明结构体类型的指针变量，这样的话就可以让指针在结构体数组上移动。结构体中有一个叫做 NEXT 的指针类型元素，它指向结构体数组的下一个元素首地址，这样一串结构体就被串连在一起，形成“链表”。一串结构体的最后一个变量 NEXT 成员指向 NULL，即空指针，代表一个链表的结束。链表属于数据结构部分的内容，不在这里详细讲解，有兴趣可以查阅参考资料 1 的教材内容。

五、 枚举类型 enum

当一个变量只有几种值可以选择，这种变量称为枚举类型。声明枚举类型的关键字是 enum，变量的值域只在列出的值之中。例如：

```
1. enum  Week{Sun , Mon , Tue , Wed , Thu , Fri , Sat};
```

这个时候我们定义了一个枚举类型 Week，我们就可以用这个类型来定义变量：

```
1. enum  Week  Sep_1st , Oct_2nd;
```

这个时候 Sep_1st 和 Oct_2nd 都是 Week 类型的变量了，它们的值只能是已定义的 Week 类型的值其中之一。声明枚举类型的一般形式为：

```
1. enum  枚举名  {枚举元素列表};
```

六、 其他的自定义类型

除了结构体和枚举类型之外，我们还有共用体类型没有详细讲解，可以查看参考资料 1 的教材自行了解。

我们还可以使用 typedef 来指定新的类型名来代替已有类型名。例如：

```
1. typedef  int  Zheng;
```

即指定 Zheng 这个类型名来代替 int。这个方法同样适用于重新定义结构体类型等。

七、 练习题

1. 建立一个学生通讯录，可以用户自行输入数据，然后全体输出。
2. 定义一个结构体变量，包含年、月、日三个元素。然后设计一个函数可以计算这天在这一年中是第几天。注意闰年的问题。

第八部分：附录

附录 1：ASCII 码对照表

Bin	Dec	Hex	缩写/字符	解释
00000000	0	00	NUL(null)	空字符
00000001	1	01	SOH(start of headling)	标题开始
00000010	2	02	STX (start of text)	正文开始
00000011	3	03	ETX (end of text)	正文结束
00000100	4	04	EOT (end of transmission)	传输结束
00000101	5	05	ENQ (enquiry)	请求
00000110	6	06	ACK (acknowledge)	收到通知
00000111	7	07	BEL (bell)	响铃
00001000	8	08	BS (backspace)	退格
00001001	9	09	HT (horizontal tab)	水平制表符
00001010	10	0A	LF (NL line feed, new line)	换行键
00001011	11	0B	VT (vertical tab)	垂直制表符
00001100	12	0C	FF (NP form feed, new page)	换页键
00001101	13	0D	CR (carriage return)	回车键
00001110	14	0E	SO (shift out)	不用切换
00001111	15	0F	SI (shift in)	启用切换
00010000	16	10	DLE (data link escape)	数据链路转义
00010001	17	11	DC1 (device control 1)	设备控制 1
00010010	18	12	DC2 (device control 2)	设备控制 2
00010011	19	13	DC3 (device control 3)	设备控制 3
00010100	20	14	DC4 (device control 4)	设备控制 4
00010101	21	15	NAK (negative acknowledge)	拒绝接收
00010110	22	16	SYN (synchronous idle)	同步空闲
00010111	23	17	ETB (end of trans. block)	传输块结束
00011000	24	18	CAN (cancel)	取消
00011001	25	19	EM (end of medium)	介质中断
00011010	26	1A	SUB (substitute)	替补
00011011	27	1B	ESC (escape)	溢出
00011100	28	1C	FS (file separator)	文件分割符

00011101	29	1D	GS (group separator)	分组符
00011110	30	1E	RS (record separator)	记录分离符
00011111	31	1F	US (unit separator)	单元分隔符
00100000	32	20	(space)	空格
00100001	33	21	!	
00100010	34	22	"	
00100011	35	23	#	
00100100	36	24	$	
00100101	37	25	%	
00100110	38	26	&	
00100111	39	27	'	
00101000	40	28	(	
00101001	41	29	)	
00101010	42	2A	*	
00101011	43	2B	+	
00101100	44	2C	,	
00101101	45	2D	-	
00101110	46	2E	.	
00101111	47	2F	/	
00110000	48	30	0	
00110001	49	31	1	
00110010	50	32	2	
00110011	51	33	3	
00110100	52	34	4	
00110101	53	35	5	
00110110	54	36	6	
00110111	55	37	7	
00111000	56	38	8	
00111001	57	39	9	
00111010	58	3A	:	
00111011	59	3B	;	
00111100	60	3C	<	
00111101	61	3D	=	
00111110	62	3E	>	
00111111	63	3F	?	
01000000	64	40	@	
01000001	65	41	A	
01000010	66	42	B	
01000011	67	43	C	
01000100	68	44	D	
01000101	69	45	E	
01000110	70	46	F	
01000111	71	47	G	

01001000	72	48	H	
01001001	73	49	I	
01001010	74	4A	J	
01001011	75	4B	K	
01001100	76	4C	L	
01001101	77	4D	M	
01001110	78	4E	N	
01001111	79	4F	O	
01010000	80	50	P	
01010001	81	51	Q	
01010010	82	52	R	
01010011	83	53	S	
01010100	84	54	T	
01010101	85	55	U	
01010110	86	56	V	
01010111	87	57	W	
01011000	88	58	X	
01011001	89	59	Y	
01011010	90	5A	Z	
01011011	91	5B	[	
01011100	92	5C	\	
01011101	93	5D	]	
01011110	94	5E	^	
01011111	95	5F	_	
01100000	96	60	`	
01100001	97	61	a	
01100010	98	62	b	
01100011	99	63	c	
01100100	100	64	d	
01100101	101	65	e	
01100110	102	66	f	
01100111	103	67	g	
01101000	104	68	h	
01101001	105	69	i	
01101010	106	6A	j	
01101011	107	6B	k	
01101100	108	6C	l	
01101101	109	6D	m	
01101110	110	6E	n	
01101111	111	6F	o	
01110000	112	70	p	
01110001	113	71	q	
01110010	114	72	r	

01110011	115	73	s	
01110100	116	74	t	
01110101	117	75	u	
01110110	118	76	v	
01110111	119	77	w	
01111000	120	78	x	
01111001	121	79	y	
01111010	122	7A	z	
01111011	123	7B	{	
01111100	124	7C	|	
01111101	125	7D	}	
01111110	126	7E	~	
01111111	127	7F	DEL (delete)	删除

附录 2：C 语言中的关键字

auto	break	case	char	const
continue	default	do	double	else
enum	extern	float	for	goto
if	inline	int	long	register
restrict	return	short	signed	sizeof
static	struct	switch	typedef	union
unsigned	void	volatile	while	bool
_Complex	_Imaginary			

附录 3：参考文献：

[1]谭浩强〔M〕. C 程序设计（第四版），清华大学出版社

第九部分：后记

其实就是前言，因为前言部分一直空缺着，一直到所有内容都写完了才回来写，到了最后发现写的前言有点多，就全都放到后记来了。这个结构正如 C 语言的学习，一开始摸不着头脑，是因为 C 语言是一个整体，当模模糊糊地学完了全部，再一结合，才能明白其中各部分互相的联系。

这个文档在编写时参考了谭浩强老师的《C 程序设计》一本书，一来这本书是我们专业的教材，二来是喜欢这本书，喜欢谭浩强老师的写书风格。浅显易懂，言简意赅。夸谭老师的话在书的前 16 页都夸完了，也不用我在这里费口舌。

这个文档起稿于 2015 年 9 月末，第一版完成于 2015 年 10 月 6 日，至于 BUG 还没有找全，也没有调试过所有出现的例程。万事总有疏漏之处，正因如此我们才一直都在进化。

这个文档写给大连工业大学集成测控技术研究所 MOS（微/移动操作系统）小组 2015 届的新生们，同时送给以后每一届的学弟学妹。写这个文档的动机很

纯洁——我最喜欢的谭浩强老师的书实在是太繁杂了，尽管语言通俗易懂，内容还是需要更加精炼。一句话就能说完的话一定不要说两句。准确的说，这个文档是谭浩强老师教材的精简版。写给以后每一届学弟学妹，希望各位能重视起 C 语言这一门课程。尽管它在你们的心目中完全不是一门编程语言，因为学了几周也写不出一个自己的窗口（UI），人家学 Java 的一天就能出现一个可视化的窗口。不论怎样，记住，C 语言永远都是编程的根本。C 语言培养的是一个人的编程思想，顺带学一些编程的基础。没有这个基石，以后学什么语言、写出什么高大上的程序，都只是空中楼阁。

至于 C 语言的学习，看完这个文档还不够，这只是一个入门教程。本文档里没有详细介绍链表、共用体、文件的相关操作等内容。这里的内容用来入门是足够了，但想要进阶还是很难。今后有的同学还要考国家计算机二级，就要学更多的一些内容再做更多的习题。

这个文档配合 2015 年新生考核期的 C 语言培训使用，预计 2 周。