*第二章 数据的容器——变量*

第一节：条件运算符

第二节：常量

第三节：变量的运算

第四节：格式化输入输出

第五节：其它输入输出函数

2.1变量类型与保留字

前面1.3节展示了一个最简单的C语言程序，它只具有输出的功能。而事实上，大多数的程序都不可能只是输出几句话这么简单。一个具有强大功能的程序需要访问大量的数据，而这些数据中则是用变量（**variable**）来保存的。如果把数据比作货物，那么变量就是货物的包装盒，用来容纳数据。C语言的变量和数学上的变量有些类似，都可以理解为“能够改变的量”。变量十分常见，比如某个时刻的气温、一片草地中小动物的数量、甚至某个人银行卡中的存款，这些都可以视为变量。

我们知道，现实中的容器分为很多种，每种容器的用途与大小不尽相同。同样地，C语言中的“容器”也有类型与大小之分。比如，储存两个整数的和需要用“整数型”的变量，储存字符应该用字符型变量。下表将列出常见的变量类型（并非所有）。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **主类型** | **分类型** | **修饰符** | **占用内存** | **取值范围** |
| 整形int | 短整型 | short | 2Byte | -32768~32767 |
| 长整形 | long(可省去) | 4Byte | -231~231-1 |
| 无符号短整型 | unsigned short | 2Byte | 0~65535 |
| 无符号长整形 | unsigned long | 4Byte | 0~232-1 |
| 实型 | 浮点型 | float | 4Byte | -3.4e38~3.4e38 |
| 双精度型 | double | 8Byte | -1.7e308~1.7e308 |
| 字符型 char | 有符号型 | signed(可省去) | 1Byte | -128~127 |
| 无符号型 | unsigned | 1Byte | 0~255 |
| 布尔型 \_Bool | 无 | 无 | 1Byte | true false |

表2-1-1

其中，布尔型数据是C99以后的C语言才有的，0表示false，非0表示true。

想要“制造”一个容器，不仅要知道它是什么类型的，还要知道它叫什么名字。起名是有很多讲究的，其中，最重要的一点就是变量名不能为C语言的保留字（**reserved word**）。所谓保留字，是指已经被C语言“使用”的名字，它们有着特定的含义，如果用它们做变量名，就会引起混淆，这和人类起名时遵循的规范类似。

C语言的保留字如下表所示：

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| asm | auto | break | case | char | const | continue | defualt | do |
| double | else | enum | extern | far | float | for | goto | if |
| int | long | near | register | return | short | signed | sizeof | static |
| struct | switch | typedef | union | unsigned | void | volatile | while |  |

表2-1-2

千万不要尝试现在就记忆这两个表格，而是应该在循序渐进的学习中逐步理解它们，否则只会事倍功半。

从表2-1-1中可以看出，C语言的变量类型分为几大类。接下来我们将分析每种类型分别有什么特点。

整形变量用来保存整数，在声明一个整形变量的时候，默认是long型，在需要的时候可以用其它修饰符来修饰。每种分类型都有其使用价值。整形数据不能保存小数，但是整形数据是精确的，不会出现四舍五入的情况。

实型变量分为浮点型和双精度型，二者的区别在于占用空间的大小与表示范围。C语言中的实型变量有两种表示方式，一种是小数形式，比如3.14、6.02等等。还有一种方式就是指数形式，也就是表2-1-1中使用的方式，比如1.7e308，就等价于1.7×10308，e后面的数字必须是整数。

字符变量在储存字符的时候，本质上也是通过数字来储存的（字符与数字之间的转换依据为ASCII码，字符的ASCII码表请参见附录1）。字符变量可以储存一个半角的西文字符或者一个转义字符（要注意的是，字符型数据在表示的时候需要加上单引号），转义字符是指键盘上没有或者不方便直接表示的字符，下表将给出一些常见的转义字符：

|  |  |
| --- | --- |
| 转义字符 | 功能 |
| \a | 响铃 |
| \n | 换行 |
| \t | 相当于Tab键 |
| \’ | 单引号 |
| \” | 双引号 |
| \? | 问号 |
| \\ | 反斜杠 |

表2-1-3

在上表中，响铃和回车是键盘上没有的。Tab则是不方便表示的（Tab产生的结果只是将光标跳到下一个制表位置）。而单引号、双引号、反斜杠、问号有着其它的作用，直接使用将会触发其它的作用，而不能被正常表示，现在看不懂没关系，以后的学习中会逐步加深对它们的理解。

2.2变量的命名与初始化

前面一节介绍了变量的类型，这一节将讲解如何“制造”一个变量。变量“制造”的过程称为变量的声明（**declare**），要想使用一个变量，必须在使用之前声明它。

变量的声明格式是：变量类型 变量名;（分号必须带上）

如：int a;、char b;等等。如果要声明多个同类型的变量，需要用逗号隔开，如：int a,b,c;

2.1节说到，变量命名是有讲究的，就好像人类起名也必须遵守一定的规则一样，变量的命名应符合以下的规则：

1. 变量名不能是保留字。
2. 变量名只能包含字母、数字和下划线，就像中国人起名只能是汉字，不能是带有ABC或者其它特殊符号。
3. 变量名的第一个字符必须是字母或下划线（不能是数字）。
4. 变量名的字母的大小写是敏感的，对应的大写字母与小写字母被视为不同字符。

此外，变量名还需要符合它所储存的数据的特征。比如说存放学生数量的int型变量就最好命名为numOfStu。这样命名的好处在于阅读代码时能一目了然。试想，如果有一个很长的程序，其中所使用的变量非常多，如果对每个变量都用a,b,c这样的命名方式，那么每个变量的作用就很难被体现出来，这样大大减少了程序的可读性和可维护性。当然，程序足够简单时，用a,b,c也是可以的，但是一定不能养成习惯，否则很难改正。

在我们“制造”了一个容器之后，可能马上就需要使用它，也可能暂时不用。如果马上就需要使用，需要对它进行“初始化”，初始化的格式为：变量类型 变量名=初始值;如：int a=10;、char b=’b’;等，如果没有对变量进行初始化，那么变量储存的数据实际上是不确定的（可能这个容器并不是新制造的，而是原来用过的，容器里面还有其它东西）。=在C语言中是赋值运算符，而不同于数学中的等号。赋值运算符在2.4节会被详细介绍。

**小提示**

在声明变量的时候，应该注意节约资源。比如要保存学生的数量，用double型数据实在显得浪费，虽说浪费的这点内存对于当今的高性能电脑来说微不足道，但是当这种浪费积累到一定程度之后（比如大型软件），就会大量消耗计算机的性能。

2.3常量

如果说变量是可以改变的量，那么常量（**constant**）就是“经久不变的量”。常量也是很普遍的，比如圆周率π、阿伏伽德罗常数NA、光在真空中的速度c等等。这些量不会发生改变，也不能被改变。

声明一个常量的语法格式与声明变量类似，但是要在数据类型前加上“const”，并且必须对其进行初始化。比如 const int a=10;、const char b=’b’ 在声明了常量之后，就不能再改变常量的值，否则会导致错误。

另外，还有一种通过“宏定义”来“声明”常量的方法。前面1.3节介绍过，“#include <stdio.h>”表示在预编译的时候包含stdio.h这个头文件。其实预编译不只可以包含头文件，还可以定义一些常量。

宏定义的语法格式为：#define 常量名 初始化值（不加分号）

比如#define A 10 就相当于声明了一个常量A，A的值为10。

2.4变量的运算

变量的运算种类是非常多的，本节只介绍其中相对基础的部分：赋值运算、算术运算、关系运算以及变量的类型转换，更复杂的运算符会在后面章节中被提及。

前面2.2节说到，=并不是是等号，而是一种赋值运算符。其语法格式为：

左值=表达式;

其中，左值（**left value**）是变量，表达式则可以是常量或者变量甚至是其它运算表达式。假设有一个整形变量a，对a进行如下操作：

a=10;

a=5;

这个过程相当于是先把10赋值给a，然后把5赋值给a,两次运算之后，a的值变成了5（而不是10）。

算术运算符包括四则运算（加减乘除）、取余运算、自增自减运算。

四则运算的运算符分别是：+、-、\*、/。乘号和除号与数学中有所不同。一般四则运算都与赋值运算有关，比如a=10+1;就是把10和1相加，然后赋值给a。当然，参与四则运算的不止可以是常量，也可以是变量。

取余运算只能发生于两个整形的量（变量或常量）之间。取余运算符为%。比如10%3的结果是1，也就是用10来除以3，结果为余数1。

自增和自减运算符分别是++和--，如a++;、a--;它的本质是a=a+1;或a=a-1;也就是将a+1或a-1的值赋值给a。而自增自减运算符可以放在变量前面，也可以放在变量后面。两种方法对于变量本身来说产生的结果是一致的，但是如果此变量同时参与了其它运算，那么就会有一点区别。比如a=b++;的含义是先把b的值赋给a，然后再自增，而a=++b;则是先将执行自增，然后再把值赋给a。

其实赋值运算符不止有=一种。还有更复杂的复合算术赋值，比如+=、-=、\*=、/=。它们的意义也并不复杂，比如a+=b;等效于a=a+b;、a/=b;等效于a=a/b;。

除了赋值运算和算术运算，关系运算也是很常见的。关系运算符用来做比较运算，有这么几种：

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **运算符** | > | < | == | >= | <= | != |
| **含义** | 大于 | 小于 | 等于 | 大于等于 | 小于等于 | 不等于 |

表2-4-1

关系运算的结果为一个布尔型数据，如3>2的结果为true，3<2的结果为false。

在赋值运算中，可不可以把double型数据赋值给int型变量，或者将int型数据赋值给double型变量呢？答案是可以的，只是可能导致数据变得更加不精确。

生活中，不是所有的容器都用来盛放它本来应该容纳的东西。有时候，我们需要把笔放进裤子口袋里，这时裤子的空间绰绰有余；有时候我们还需要把一大堆杂物放进一个小盒子里，很明显这时候空间就不够了，我们就不得不舍掉一些不必要的物品，把“主干”放进盒子。

我们知道，double型是可以表示整数的，但是int型却只能表示浮点数的整数部分。如果我们非要用double型来表示整数，除了浪费空间以外，没有任何问题，但是如果我们试图用int型来表示一个浮点数，就不得不把小数部分去掉。

C语言中这种类似于用双精度型数据来表示整型数据被称为类型转换。类型转换的语法格式如下：

(目标类型) 数据名

比如有一个double型的数据a，我们需要把它的整数部分赋值给int型变量b，就应该这么写：b=(int)a;

要注意，在类型转换的过程中，a的数据类型是不会发生改变的，它依然是双精度型。

2.5格式化输入输出

绝大多数的程序或软件都不是在后台就解决了所有的事，事实上，要解决问题，需要大量的输入与输出。C语言中的格式化输入输出是最基本的人机交互，没有了输入与输出，就很难写出有强大功能的程序。

我们在1.3节知道了输出要用到printf函数，接下来我们将详细介绍printf函数的基础用法。

printf(“Hello World!”);是最简单的输出，只需要改变双引号内的内容就能实现输出不同的字符串，这种输出称为按原样输出，在双引号内的内容，除了一些有“特殊含义”的字符以外，都是按原样输出的。转义字符是一种特殊字符，另外还有一类特殊字符叫格式控制字符。格式控制字符的意义何在呢？

如果有一个int形变量a，我们想要输出a该怎么办呢？可能有的读者会认为这样就行了：printf(“a”);事实上，这样是不行的。

既然是格式化的输出，首先我们要告诉系统，我们要输出的是整数、实数，还是字符。这时候就要用到格式说明符，格式说明字符总是带有%，比如%d、%c等等。下标给出部分常用的格式字符：

|  |  |
| --- | --- |
| 格式控制字符 | 作用 |
| %d | 将一个数据按整数的格式输出 |
| %f | 将一个数据按浮点数的格式输出 |
| %c | 将一个数据按字符的格式输出 |
| %s | 将一个数据按字符串的格式输出 |
| %p | 将一个数据按地址的格式输出 |

表2-5-1

可以看出，要想输出int型的变量a，首先要用到%d这个格式控制字符，所以输出的时候至少需要这么写：printf(“%d”);，可是，这样写还是有问题的，因为系统只知道要输出一个int形数据，这个int型数据是什么系统却不知道，所以我们应该告诉系统，这个int形数据是a。于是正确的写法是这样：printf(“%d”,a);。这样，在输出的时候，系统遇到第一个%d之后，就会到双引号之后去寻找%d对应的参数，找到的是a，于是就把a的值按整数的格式输出了。

如果想要用printf输出多个变量怎么办呢？假设要同时输出整形变量a和b，对应的语句就是printf(“%d%d”,a,b);。前面两个%d表示要输出两个整形数据，而这两个整型数据是a和b。

来看一个代码演示：

1. #include <stdio.h>
2. int main()
3. {
4. double a=1.25;
5. printf("%f %.1f %d\n",a,a,a);
6. return 0;
7. }

程序2-5-1

运行结果为：1.250000 1.3 0

在上面这段代码中，声明了一个双精度型的变量a，并初始化为1.25。分别按浮点型、保留一位小数、整型输出。如果直接按浮点型输出，系统将默认有6位小数，当我们指定了保留一位小数之后，系统就只显示一位小数，多出的部分四舍五入。而按整形输出的时候，由于类型不吻合，所以并没有输出a的整数部分，而是输出了0，如果想要输出整数部分，需要进行类型转换。

**想一想**

如果要输出一个字符型变量，应该用什么格式控制字符？请亲自编写程序测试

仅仅是控制输出的类型还不能算是格式化输入输出。真正的格式控制还包括对齐和长度的控制。

假设我们要让a在输出的时候右对齐，并且至少占用3个宽度，应该怎么写呢？这时候，就需要在%和d之间增加一个3：printf(“%3d”,a);如果a占用的宽度只有1（a大于0小于9），那么在输出的时候，a就会紧贴右边，而在左边留下两个空格，如果a的宽度为4，那么在输出的时候就会占用4个空间（按实际长度输出），不再用空格来占位。如果我们希望a输出的时候左对齐而不是右对齐，就应该在%后面添加这个符号：-。比如printf(“%-3d”,a);。

在输出浮点型数据时，我们还要考虑到有效数字的问题。假设有一个浮点型数据b，我们希望输出1位有效数字，并且输出结果右对齐3个宽度，这时候就需要用这个字符来控制：.1，它的位置在控制对齐的数字的后面：printf(“%3.1f”);。当然，不想要右对齐的话，把3去掉即可。

完整的printf格式为：

**printf(“格式控制字符”,输出列表);**

printf的使用技巧非常多，为了降低学习的难度，这里只提到了最核心的部分，有兴趣的读者可以在有一定的基础之后，再去查阅相关资料。

输出时用的函数是printf，而输入的时候则是用scanf函数。scanf函数的格式为：

**scanf(“格式控制字符”，参数地址列表);**

scanf函数与printf函数很像，但是它们之间也是有差别的，比如要输出一个int形变量a，用的是printf(“%d”,a);，但是在输入的时候却要写成scanf(“%d”,&a);。多出来的这个&是2.4节没有提到的操作符，它叫做取地址操作符。我们知道a是保存在内存中的某个地方的，&a就表示a在内存中的地址。得到了地址，scanf就能为a赋值了。

**想一想**

如果要输入一个char型变量，应该怎么编写语句？

假设我们想要用户输入一个三位数赋值给整形变量a，但是用户在输入的时候可能不小心多输入了一位数，这时候就达不到预期的效果。正确的语句应该是这样：scanf(“%3d”,&a);，这里的3是限制最大输入长度的，当输入长度小于3的时候，按实际长度输入，当输入长度大于3的时候，只截取3位数。这一点和格式化输入很类似，只是要注意，在输入的时候，是不可以控制输入“有效数字”的，也就是说，形如scanf(“%3.1d”,&a);的写法是错误的。

如果要一次性输入多个数据呢？在输入的时候怎么区分它们？scanf函数，还有一个相关概念叫做分隔符，所谓分隔符，就是指在输入的时候分隔每个数据的字符，scanf函数并不会读取这些字符，而是在遇到这些字符时视为分隔的标志。分隔符包括‘\t’、‘\n’以及空格符。比如scanf(“%d%d”,&a,&b);，如果我们输入1空格2，那么1就会被赋值给a，2会被赋值给b。

此外，如果想要输入一个双精度型的变量b，需要用lf来作为格式控制字符，如：scanf(“%lf”,&b);。

scanf的内容同样是非常多的，为了降低学习难度，我们依然只介绍最核心的部分。

scanf函数读取数据的方式缓冲读，接下来的学习中我们会逐渐了解到它的机制。

接下来通过一个程序来演示怎么输入一个双精度型数据：

1. #include <stdio.h>
2. int main()
3. {
4. double a;
5. printf("请输入一个浮点数：\n");
6. scanf("%lf",&a);
7. printf("刚才输入的是%.1f\n",a);
8. return 0;
9. }

程序2-5-2

在这段代码中，第6行的作用是输入一个double型变量，如果不把格式控制字符写成%lf，就不能正确地输入双精度型数据。

**试一试**

声明一个int形变量a。提示用户输入一个数，并把这个数赋值给a。要求不管用户输入了几位数，都只读取前3位。

2.6字符的输入输出函数

在输入输出字符的时候，用scanf和printf函数是没问题的，但是也可以用其它的函数来实现。

如果是输出单个字符，可以用putchar函数，如：

putchar(‘a’);表示输出字符’a’。

putchar(a);则表示输出char型变量a中所储存的数据（不一定是字符‘a’）。

而输入单个字符可以用到getchar函数，如：

a=getchar();

要注意，a应该写在=的左边，而不是像putchar(a)一样写在括号内部。

要想使用getchar和putchar函数，需要包含stdio.h头文件(在程序开头写上#include <stdio.h>)。

getchar函数和scanf函数都是属于缓冲读。用户在输入字符的过程中，输入的数据首先被保存在一个叫键盘缓冲区的地方，只有在按下回车键之后，getchar函数才会开始从这个缓冲区中读取数据,不管缓冲区中有多少数据，getchar函数都只会读取第一个。

缓冲读数据的方式是有好处的。如果当我们按下一个键的时候程序就读取我们输入的信息，那么在输入多位数的时候就很尴尬了。但是缓冲读也有缺点，第一是非要等我们按下回车键程序才会有所响应，第二是缓冲读可能造成部分数据残留在缓冲区中，下次读入数据时就会优先读取残留的数据，从而造成干扰。

\_getch函数和\_getche函数是非缓冲读数据的，当我们按下一个键之后，程序立马就会响应，不管按下的是否是回车键。

想使用\_getch和\_getche函数首先要预编译conio.h头文件，要注意的是，conio.h并不是C语言的标准头文件，所以并不适用所有编译器，但是在Visual Studio上是可以使用的。

\_getch和\_getche的用法很类似：

a=\_getch();

a=\_getche();

二者唯一的区别在于通过\_getche输入的字符会在程序窗口中被显示，而通过\_getch函数输入的字符则不会被显示。

前面1.3节中，我们说“启动”目标程序的快捷键是Ctrl+F5的组合键。之所以用这个组合键，是为了暂停程序观察结果，事实上，这个暂停的功能是编译器提供的，当我们找到并执行工程文件夹里真正的可执行文件(.exe)的时候，程序会发生“秒退”的现象。系统并不会为我们平时使用的各种程序提供暂停的效果，所以我们需要自己来完成暂停的功能。getchar和getch函数正好可以起到起到暂停函数的作用。通常情况下，getch函数都能完成暂停程序的作用，但是getchar函数却会“灵时不灵”，接下来通过一个程序来展示怎么通过这两个函数来暂停程序，并说明“时灵时不灵”的理由。

1. #include <stdio.h>
2. int main()
3. {
4. char a;
5. printf("请输入一个字符：\n");
6. a = getchar();
7. printf("输入的是%c,按任意键继续...\n",a);
8. getchar();
9. return 0;
10. }

程序2-6-1

这次，我们“启动”程序不再是通过Ctrl+F5，而是直接按F5。输入abc回车,观察程序运行结果，很明显，程序“秒退”了。

明明第9行有等待输入的getchar函数，为什么程序还是不能暂停呢？原来，getchar函数读取数据是缓冲读，当我们输入abc回车之后，只有一个数据被a读取，其它的都还残留在缓冲区中。而程序执行到第9行时，便不再等待输入，而是直接读取stdin流中的残留数据，所以程序未能暂停。

读者可自行尝试，把第9行的getchar改成\_getch，观察程序是否能实现暂停。答案当然是可以的，因为\_getch函数并不从缓冲区中读取数据，所以不管之前输入了什么，都不会影响\_getch函数的读取。

如果要让第9行的getchar函数起到暂停的效果，需要重置缓冲区，重置缓冲区的语句为：rewind(stdin);。rewind也是一个函数，想要使用它需要包含stdio.h头文件。

请再次尝试在第9行代码之前插入一行清空缓冲区的语句，并观察程序运行结果。

练习题

1.声明一个字符型变量，并初始化为小写字母a，将这个字符型变量按字符输出(提示：小写字母a的表示为’a’而不是a)。

2.声明两个整形变量，提示用户输入两个数据，然后输出这两个数的和。

3.声明两个字符型变量，先用getchar函数输入其中一个，然后重置缓冲区，再次输入另一个，接着按字符输出两个变量。并用gethchar函数暂停程序。