第 3 章

标准输入和输出

引言

在编程的世界里，信息的输入和输出是程序与用户交互的桥梁。想象一下，如果我们编写的程序是一个智能机器人，那么输入函数就是机器人的耳朵和眼睛，负责接收外界的指令和信息，而输出函数则是它的嘴巴和显示器，负责将处理后的结果或信息反馈给用户。

在 C 语言中，输入输出函数为我们提供了这样的能力。通过它们，我们可以让程序从键盘读取用户输入的数据，也可以将程序运行的结果展示在显示器上，甚至还可以将数据写入文件或从文件中读取数据。

本章我们将深入学习 C 语言中的几个关键输入输出函数：scanf()、printf()、getchar()、putchar()。我们将通过实例演示这些函数的用法，并探讨它们在编程中的实际应用。

在此之前，请大家先思考一个问题：在日常生活中有没有遇到过需要使用程序来输入和输出数据的场景呢？比如，一个简单的计算器程序需要读取用户输入的数值，并显示计算结果；一个学生信息管理系统需要读取学生的信息，并将这些信息保存在文件中以便日后查阅。这些场景都涉及输入输出操作。接下来，让我们带着对输入输出操作的好奇心和探索精神，一起开始本章的学习内容吧！

本章导读

本章我们将学习 C 语言中的输入输出函数，重点掌握 scanf()函数和 getchar()函数用于从标准输入(通常是键盘)读取数据，以及 printf()函数和 putchar()函数用于向标准输出(通常是显示器)显示数据的方法。

重点内容

(1) scanf()函数的使用方法和格式控制符。

(2) printf()函数的使用方法和格式控制符。

(3) getchar()函数的使用方法。

(4) putchar()函数的使用方法。

世界计算机名人——肯尼思·莱恩·汤普森

在“C 语言程序设计”这门课程中，我们不仅要学习编程语言的基础知识和编程技巧，更要从中汲取坚韧不拔、勇于创新的精神力量。今天，我们将通过讲述计算机科学史上的一位传

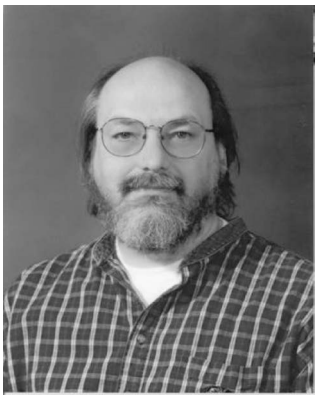

图 3-1 肯尼思·莱恩·汤普森

奇人物——肯尼思·莱恩·汤普森(Kenneth Lane Thompson)(见图 3-1)的故事,来激励同学们在学习和生活中不断追求卓越,坚持不懈。

早年经历与自学成才

肯尼思·莱恩·汤普森 1943 年出生于美国新奥尔良。他从小就对逻辑和数学有着浓厚的兴趣,上小学时便对二进制产生了浓厚的兴趣,中学时更是自己动手制作无线电、示波器和放大器。这种对技术的热爱和执着,为他日后的成就奠定了坚实的基础。

1960 年,肯尼思·莱恩·汤普森考入加利福尼亚大学伯克利分校,主修电气工程。在校期间,他自学编程,并通过不断的努力和实践,逐渐成长为一名优秀的程序员。他的故事告诉我们,无论出身如何,只要有兴趣和坚持,就能在自己的领域里取得非凡的成就。

UNIX 操作系统的诞生与影响

1966 年,肯尼思·莱恩·汤普森加入贝尔实验室,开始了他职业生涯中最为辉煌的篇章。在参与 Multics 项目的过程中,他因项目被取消而决定自己开发一个操作系统来玩游戏。正是这份对技术的热爱和执着,让他在短短一个月内编写出了 UNIX 操作系统的雏形。

UNIX 操作系统的诞生不仅改写了计算机操作系统的历史,更成为软件界的“瑞士军刀”,其简洁、稳定与高效的特点赢得了业界的广泛赞誉。肯尼思·莱恩·汤普森也因此与丹尼斯·麦卡利斯泰尔·里奇共同获得了 1983 年的图灵奖,成为计算机科学领域的佼佼者。

勇于探索,敢于创新

肯尼思·莱恩·汤普森的故事告诉我们,只有勇于探索未知领域,敢于挑战传统观念,才能在激烈的竞争中脱颖而出。在学习 C 语言程序设计的过程中,我们也要敢于尝试新的编程思路和方法,不断创新和改进自己的程序。

坚持不懈,持之以恒

成功往往属于那些能够坚持不懈、持之以恒的人。肯尼思·莱恩·汤普森在研发 UNIX 操作系统的过程中遇到了许多困难和挫折,但他从未放弃过自己的梦想和目标。同样,我们在学习和生活中也要保持这种精神状态,不断克服困难、挑战自我、实现自我超越。

团队协作,共同进步

UNIX 操作系统的成功也离不开肯尼思·莱恩·汤普森与丹尼斯·麦卡利斯泰尔·里奇的紧密合作。他们相互支持、共同努力,最终创造出了这一伟大的操作系统。在学习 C 语言程序设计的过程中,我们也要注重团队协作和相互学习,与同学们共同进步、共同成长。

3.1 输入和输出的基本概念

数据的输入和输出指计算机主机和外设之间的数据流通。具体而言,输入意味着使用输入设备,如键盘、鼠标、麦克风或扫描仪,将数据导入计算机内部。而输出,则是将存储在

计算机内部的数据传输到输出设备，如显示器、打印机或磁盘，以呈现处理结果。输入为计算机提供了待处理的数据基础，而输出则负责将处理后的信息以可视化的方式展示在显示器上。

C 语言本身并不包含专门的输入和输出语句，而是依赖于 C 语言标准库中的函数来执行这些操作。这些标准函数被组织成库，并在 C 系统中存储。这些函数的名称并不是 C 语言的关键字，而是作为普通标识符来使用的。C 语言标准输入和输出标准函数如表 3-1 所示。

表 3-1　C 语言标准输入和输出标准函数

函　数　名	功　　能
printf()	用于将格式化的信息输出到显示器上，一般形式为 printf("格式控制字符串", 输出列表)
scanf()	用于从键盘上接收输入，并将输入的值赋给变量，一般形式为 scanf("格式控制字符串", 地址列表)
getchar()	用于从键盘上接收一个字符
putchar()	用于向显示器输出一个字符
scanf_s()	在接收输入时会进行参数边界检查
printf_s()	在向显示器输出时会检查格式控制字符串是否有效

需要注意的是，除了表 3-1 列示的这些输入和输出函数，还有输入和输出一个字符串的函数 gets()和 puts()，从文件读取和写入字符串的函数 fgets()和 fputs()，从文件中进行标准化读取和写入的函数 fscanf()和 fprintf()等，这些将在后续章节中学习。

为了使用这些输入和输出的标准库函数，需要通过预编译指令 # include 将头文件 stdio. h 包含到源代码文件中。即程序开头应有如下预编译命令。

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

这个头文件位于程序的开头，作用是提供与标准输入和输出相关的变量定义、宏定义以及函数的声明。

stdio. h 是“standard input & output header”的缩写，意为标准输入输出头文件。它包含了用于执行输入和输出操作的各种函数，这些函数允许程序与用户进行交互，读取用户的输入，并显示程序的输出结果。通过使用这些标准库函数，C 语言提供了强大而灵活的输入输出功能。

3.2　字符型常量

C 语言中的字符型常量是用一对单引号引起来的单个字符。字符型常量在内存中占 1 字节，存储的是该字符的 ASCII 值。

3.2.1　ASCII 字符集

在所有字符集中，最知名的可能要数 ASCII 的 8 位字符集了。ASCII 是基于拉丁字母的一套计算机编码系统，主要用于显示现代英语和其他西欧语言。它是最通用的信息交换

标准,并等同于国际标准 ISO/IEC 646。ASCII 使用指定的 7 位或 8 位二进制数组合来表示 128 或 256 种可能的字符。标准 ASCII 也叫基础 ASCII,使用 7 位二进制数(剩下的 1 位二进制为 0)来表示所有的大写和小写字母、数字 0～9、标点符号、在美式英语中使用的特殊控制字符。

由于标准 ASCII 规定的字符数目有限,在实际应用中,特别是计算机通信中,对扩展的 ASCII 的需求就显得非常强烈。扩展 ASCII 是由 ASCII 衍生出来的,它依旧使用 8 位二进制数,这样就有了 256 个不同的字符,以满足更多语言字符的需求。

3.2.2 UTF-8 字符集

ASCII 针对拉丁字母设计,随着互联网的普及和全球化发展,人们需要一种能够支持多语言环境的字符编码方式。UTF-8(Universal Character Set/Unicode Transformation Format,8 位元)的提出正是为了满足这一需求,使得在全球范围内进行信息交流和处理变得更为方便和高效。UTF-8 是针对 Unicode(统一码)的一种可变长度字符编码。它可以表示 Unicode 标准中的任何字符,并且其编码中的第一字节与 ASCII 编码相兼容,这意味着原来处理 ASCII 字符的软件不需要或只需要进行少量修改,就可以继续使用 UTF-8 编码。

UTF-8 编码具有以下主要特征。

(1) 可变长度编码。UTF-8 编码使用 1～4 字节来表示一个字符,根据具体的字符编码范围决定使用的字节数。ASCII 字符使用 1 字节,而其他 Unicode 字符则可能需要 2～4 字节。

(2) 向后兼容。UTF-8 编码对 ASCII 字符集是向后兼容的,也就是说,UTF-8 编码的文本可以正确地解码为 ASCII 文本。

(3) 多语言支持。UTF-8 编码能够编码绝大部分已知语言的字符,如英语、汉语、法语、希伯来语、阿拉伯语等。

(4) 字节序列无关。UTF-8 编码不受字节序的影响,可以在大、小端序的系统上正确解码。

(5) 存储效率高。UTF-8 编码的可变长度特性,有助于节省存储空间,特别是对于像英文字母这样占用较少字节的字符,可以提高存储和传输效率。

UTF-8 编码被广泛应用于互联网和计算机系统中,确保数据的正确性,避免兼容性问题,并节省存储空间。例如,在开发互联网应用程序时,处理用户提交的表单数据通常需要将其转换为 UTF-8 编码,以确保数据的正确性。

3.3 单个字符的输入和输出函数

C 语言中,单个字符的输入常用 getchar()函数,输出常用 putchar()函数。

3.3.1 字符输入函数 getchar()

getchar()函数是 C 语言标准库中的一个函数,用于从标准输入(通常是键盘)读取一个字符。这个函数不需要任何参数,并且返回读取到的字符。如果发生错误或者到达文件末

尾，它将返回 EOF(End Of File)，通常被定义为－1。

【例 3-1】　使用 getchar()函数读取一个字符。

```
#include <stdio.h>
int main()
{
    char ch;
    printf("请输入一个字符: ");
    ch = getchar();
    printf("你输入的字符是 %c\n", ch);
    return 0;
}
```

在这个例子中，程序首先输出一个提示信息，然后使用 getchar()函数等待用户输入一个字符。输入的字符被存储在变量 ch 中，并随后被输出出来。

运行结果如下。

```
请输入一个字符: a↙
你输入的字符是 a
```

需要注意的是，getchar()函数在读取字符后会在输入流中留下一个换行符(如果用户按回车键的话)。这可能会影响到后续对 getchar()函数或其他输入函数的调用，因为它们可能会立即返回这个换行符，而不是等待用户输入新的字符。为了避免这种情况，可以使用 getchar()函数多次，直到遇到非换行符的字符，或者使用其他函数(如 scanf())来读取和丢弃换行符。

此外，getchar()函数是非阻塞的，这意味着它会立即返回可用的字符(如果有的话)，而不会等待用户输入。这可能会导致程序在没有输入的情况下继续执行，这在某些情况下可能是用户不希望的。在这种情况下，可以使用其他函数(如 fgets())来替代 getchar()函数，以便在没有输入时阻塞程序。

由于 getchar()函数是从输入缓冲区读取字符，因此如果在之前的输入中留下了换行符或其他字符，getchar()函数可能会立即返回这些字符，而不是等待用户输入新的字符。为了避免这种情况，可以在调用 getchar()函数之前使用其他函数(如 scanf())来清除缓冲区中的换行符。

3.3.2　字符输出函数 putchar()

putchar()函数是 C 语言标准库中的一个函数，用于将单个字符输出到标准输出设备(通常是显示器)。这个函数定义在 stdio.h 中，并接收一个 int 型的参数(尽管它通常用于输出 char 型的字符)，这个参数是用户想要输出的字符的 ASCII 值。

【例 3-2】　输出字符 A。

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

```
{
    char ch = 'A';      // 定义一个字符变量并初始化为 'A'
    putchar(ch);        // 输出字符 A
    putchar('\n');      // 输出换行符,以便在终端上开始新的一行
    return 0;
}
```

在上面的代码中,putchar(ch) 会将变量 ch 中存储的字符(在这个例子中是 'A')输出到显示器。然后,putchar('\n') 输出一个换行符,使得显示器的输出在新的一行开始。

运行结果如下。

```
A
```

用户也可以直接传递字符字面量给 putchar()函数,示例如下。

```
putchar('B');     // 直接输出字符 B
```

尽管 putchar()函数通常用于输出 char 型的字符,但由于它接受 int 型的参数,因此用户也可以传递任何 int 型的值给它。如果传递的值超出了字符的范围(即 ASCII 值超出了 0～127 的范围,对于扩展 ASCII 则是 0～255 的范围),那么输出的将是该值对应的非打印字符。对于非打印字符,其输出行为可能取决于用户的终端或控制台如何解释这些字符。

第 6 集
微课视频

需要注意的是,putchar() 函数只输出一个字符,并且不会自动在字符后面添加换行符。如果用户希望在字符后面有换行符,需要显式地使用 putchar('\n')或其他方法来添加。

3.4 格式输入和输出函数

在 C 语言中,格式输入输出函数主要指的是 scanf()和 printf()函数。这两个函数都定义在头文件 stdio.h 中,用于在控制台上输入和输出格式化的数据。

3.4.1 格式输入函数 scanf()

scanf()函数是 C 语言中用于从标准输入(通常是键盘)读取格式化输入的函数。它根据指定的格式从输入流中读取数据,并将这些数据存储在变量中。scanf()函数定义在头文件 stdio.h 中。

scanf()函数的基本语法格式如下。

```
int scanf(const char * format, …);
```

其中,format 是一个格式字符串,它指定了输入数据的类型和格式。“…”表示可以有任意数量的附加参数,这些参数是指向变量的指针,用于存储从输入流中读取的数据。

scanf()函数返回成功读取并赋值的变量数。如果到达文件末尾或发生输入失败，则返回值会小于预期。

格式转换说明由%开始，并以转换字符结束，用于指定各种输入值参数的输入格式，如表 3-2 所示。

表 3-2　scanf()函数的格式转换说明

格式转换说明符	用　　法
%d	输入十进制整数
%o	输入八进制整数
%x 或 %X	输入十六进制整数
%c	输入一个字符，空白字符(包括回车、空格、制表符)也作为有效字符输入
%s	输入字符串，遇到空白字符(包括回车、空格、制表符)时系统认为输入结束
%f	输入浮点数
%e 或%E	以科学记数法形式输入浮点数
%%	输入一个百分号%

此外，还可以指定宽度、精度、标志等修饰符，以更细致地控制输入格式。scanf()函数的格式修饰符如表 3-3 所示。

表 3-3　scanf()函数的格式修饰符

格式修饰符	用　　法
l	加在格式转换说明符 d、o、x、X 之前，用于输入 long 型数据；加在格式符 f、e、E 之前，用于输入 double 型数据
L	加在格式转换说明符 f、e、E 之前，用于输入 long double 型数据
h	加在格式转换说明符 d、o、x、X 之前，用于输入 long double 型数据
域宽 m(正整数)	指定输入数据的宽度，系统自动按此宽度截取所需数据
忽略输入修饰符 *	表示对应的输入项在读入后不赋值给相应的变量

【例 3-3】　使用 scanf()函数读取整数、浮点数、字符并输出。

```
#include <stdio.h>
int main()
{
    int i;
    float f;
    char c;
    // 读取一个整数
    printf("请输入一个整数: ");
    scanf("%d", &i);
    printf("你输入的整数是 %d\n", i);
    // 读取一个浮点数
    printf("请输入一个浮点数: ");
    scanf("%f", &f);
    printf("你输入的浮点数是 %.2f\n", f);
    // 读取一个字符
    printf("请输入一个字符: ");
```

```
    scanf(" %c", &c);          // 注意前面的空格,用于跳过任何空白字符
    printf("你输入的字符是 %c\n", c);
    return 0;
}
```

在例 3-3 中,%d、%f 和%c 是格式转换说明符,分别用于读取整数、浮点数和字符。这些格式转换说明符必须与对应的变量类型匹配。

运行结果如下。

```
请输入一个整数: 5↙
你输入的整数是 5
请输入一个浮点数: 5.6↙
你输入的浮点数是 5.60
请输入一个字符: A↙
你输入的字符是 A
```

注意,在读取字符时,格式转换说明符"%c"前面有一个空格。这个空格的作用是跳过任何可能的前置空白字符(如空格、制表符或换行符)。如果不加空格,scanf()函数可能会读取到用户按下回车键后留在输入缓冲区中的换行符作为字符输入。


第 7 集
微课视频

使用 scanf()函数时需要特别小心,因为不匹配的输入或意外的输入可能导致程序出错或产生不可预测的行为。例如,如果用户被提示输入一个整数,但却输入了一个字符或字符串,那么 scanf()函数可能无法正确读取数据,导致后续代码的行为出错。因此,在实际编程中,经常需要对 scanf()函数的返回值进行检查,以确保输入符合预期。

3.4.2 格式输出函数 printf()

printf()函数是 C 语言中用于格式输出的函数,它可以将指定的格式字符串以及后面的变量参数一起输出到标准输出设备(通常是显示器)。这个函数定义在头文件 stdio.h 中。

printf()函数的基本语法格式如下。

```
int printf(const char * format, …);
```

其中,format 是一个格式字符串,其中可以包含普通的字符以及格式转换说明符(以 % 开头)。"…"表示可以有任意数量的附加参数,这些参数将按照 format 字符串中的格式转换说明符进行格式化输出。

格式转换说明符用于指定如何格式化输出变量。printf()函数的格式转换说明符如表 3-4 所示。

表 3-4 printf()函数的格式转换说明符

格式转换说明符	用 法
%d 或 %i	输出十进制整数
%u	输出无符号十进制整数
%x 或 %X	输出十六进制整数

续表

格式转换说明符	用　　法
%o	输出八进制整数
%f	输出浮点数
%e 或%E	输出浮点数的科学记数法表示
%g 或%G	根据值的大小选择%f 或%e 格式输出浮点数
%c	输出字符
%s	输出字符串
%p	输出指针的地址

【例 3-4】　使用 printf()函数输出整数、浮点数、字符和十六进制整数。

```
#include <stdio.h>
int main()
{
    int a = 10;
    float b = 3.14159;
    char c = 'A';
    printf("整数: %d\n", a);                    // 输出整数
    printf("浮点数: %.2f\n", b);                // 输出浮点数,保留两位小数
    printf("字符: %c\n", c);                    // 输出字符
    printf("十六进制整数: %x\n", a);            // 输出十六进制整数
    return 0;
}
```

运行结果如下。

```
整数: 10
浮点数: 3.14
字符: A
十六进制整数: a
```

此外,和 scanf()函数一样,printf()函数还可以指定宽度、精度、标志等修饰符,以更细致地控制输出格式。printf()函数的格式修饰符如表 3-5 所示。

表 3-5　printf()函数的格式修饰符

格式修饰符		用　　法
l		加在格式转换说明符 d、o、x、X、u 之前,用于输入 long 型数据;加在格式转换说明符 f、e、E 之前,用于输入 double 型数据
L		加在格式转换说明符 f、e、E 之前,用于输入 long double 型数据
h		加在格式转换说明符 d、o、x、X 之前,用于输入 long double 型数据
宽度修饰符	—	左对齐输出(默认情况下,数据是右对齐的)
	m	指定输出字段的最小宽度。如果数据的实际宽度小于这个值,那么输出会使用空格或 0(取决于其他格式修饰符)进行填充
	*	使用动态传入的宽度。在格式字符串中,宽度值由一个后续的 int 型参数指定

续表

格式修饰符		用　法
精度修饰符	.n	对于浮点数,指定小数点后的位数;对于字符串,指定最大字符数
	*	使用动态传入的精度。在格式字符串中,精度值由一个后续的 int 型参数指定
填充修饰符	0	对于整数,如果指定了宽度且数据宽度小于指定宽度,则在左侧用 0 填充。通常与宽度修饰符一起使用
转换修饰符	+	对于带符号的整数,总是在数值前显示符号(正号或负号)
	空格	对于带符号的整数,正数前显示空格,负数前显示负号
	#	对于整数,使用替代形式输出。例如,对于八进制数,前缀 0;对于十六进制数,前缀 0x 或 0X;对于浮点数,输出包括小数点

【例 3-5】 利用格式修饰符控制 printf()函数的输出格式。

```
#include <stdio.h>
int main()
{
    int a = 123;
    float b = 123.456;
    printf("%-10d\n", a);       // 左对齐,宽度至少为 10,输出: "123"
    printf("%05d\n", a);        // 宽度至少为 5,左侧用 0 填充,输出: "00123"
    printf("%.2f\n", b);        // 保留两位小数,输出: "123.46"
    printf("%10.2f\n", b);      // 宽度至少为 10,保留两位小数,输出: "123.46"
    printf("%+d\n", a);         // 显示正数的符号,输出: "+123"
    printf("%#x\n", a);         // 十六进制输出,并显示前缀,输出: "0x7b"
    return 0;
}
```

请注意,格式修饰符通常与格式转换说明符一起使用,并且它们的顺序可能很重要。例如,在%010d 中,0 是填充修饰符,10 是宽度修饰符,而 d 是格式转换说明符(表示十进制整数)。

运行结果如下。

```
123
00123
123.46
   123.46
+123
0x76
```

需要注意如下几点。

(1) 确保 printf()函数中的格式转换说明符与后面的变量参数类型相匹配,否则可能导致未定义的行为或错误的输出。

(2) 如果使用%s 来输出字符串,请确保字符串以空字符\0 结尾,并且传递给 printf()函数的指针指向有效的内存地址。

(3) 使用浮点数格式转换说明符时,可以通过“.”后面的数字指定小数点后的位数(精度)。

(4) printf()函数返回一个整数,表示成功输出的字符数(不包括末尾的空字符)。在大多数情况下,这个返回值并不直接用于程序逻辑,但在某些情况下(如错误检查)可能是有用的。

3.5　输入输出函数的安全版本

在C语言中,scanf()和printf()是常用的输入和输出标准库函数。然而,在一些编译器和环境中,特别是那些强调安全性的环境中,为了增强安全性,提供了scanf_s()和printf_s()这两个函数分别作为scanf()和printf()的安全版本。

3.5.1　格式输入函数的安全版本scanf_s()

scanf()函数在某些情况下被认为是不安全的,主要是因为以下几个原因。

(1) 不进行边界检查。scanf()函数不会对输入数据进行边界检查。这意味着如果用户输入的数据长度超过了目标缓冲区的大小,scanf()函数会继续读取并覆盖相邻内存空间的数据,这可能导致缓冲区溢出。

(2) 字符串终止符问题。scanf()函数在读取字符串时,如果源字符串超过了目标缓冲区的大小,它不会正确地处理字符串终止符\0。这可能导致字符串在缓冲区之外仍然继续读取,从而引发安全问题。

(3) 特殊字符处理问题。scanf()函数通常将输入中的特殊字符(如空格、制表符)视为输入结束的标志,这可能导致它无法正确读取包含这些特殊字符的字符串。

(4) 难以追踪和调试。由于scanf()函数的输入长度是不可预测的,这增加了在运行时捕捉和排查潜在问题的难度。这种不确定性使得程序的脆弱性更难以发现和修复。

scanf_s()函数与scanf()函数类似,用于从标准输入设备(通常是键盘)读取并格式化数据。scanf_s()函数提供了对输入缓冲区大小的检查,从而增强了安全性。基本语法格式如下。

```
int scanf_s( const char * format, … );
```

scanf_s()函数不直接接收缓冲区大小的参数,而是在格式字符串中为每个%s、%c、%[^]等可能需要读取多个字符的转换说明符提供了一个额外的参数,用于指定目标缓冲区的大小。示例如下。

```
char buffer[50];
int result = scanf_s("%49s", buffer, (unsigned)_countof(buffer));
```

在这个例子中,%49s表示最多读取49个字符到buffer中,以留出空间给终止的空字符。_countof(buffer)是一个宏,它返回buffer数组的元素数量,这个例子中即50。

3.5.2　格式输出函数的安全版本printf_s()

printf()函数在某些情况下被认为是不安全的,主要是因为以下几个原因。

(1) 格式字符串中的漏洞。如果printf()函数的格式字符串来自不可信的输入源(例如

用户输入或网络数据),恶意用户可能会构造特定的格式字符串来触发缓冲区溢出或其他类型的攻击。这通常涉及使用%n格式转换说明符,它可以在输出中写入一个值到指定的内存地址,从而允许攻击者覆盖栈上的数据,包括返回地址,从而控制程序的执行流程。

(2) 不安全的格式化。如果printf()函数用于输出不受信任的输入数据,并且这些数据被错误地包含在格式字符串中,而不是作为单独的参数传递,那么攻击者可以通过注入特定的格式字符串和数据来操纵输出,并可能导致缓冲区溢出或其他安全问题。

(3) 未检查的参数。printf()函数不知道参数的个数,它仅仅根据格式字符串中的格式转换说明符来输出参数。这意味着如果传递给printf()函数的参数个数与格式字符串中的格式转换说明符不匹配,可能会发生未定义的行为,包括缓冲区溢出。

(4) 不安全的内存访问。printf()函数在输出字符串时,不会检查字符串是否以空字符(\0)结尾。如果传递给printf()函数的字符串没有正确终止,printf()函数可能会继续读取并输出内存中的后续数据,直到遇到空字符或遇到内存访问权限问题。这可能导致敏感数据的泄露或其他安全问题。

printf_s()函数与printf()函数类似,用于格式化输出数据到标准输出设备(通常是显示器)。不同之处在于,printf_s()函数要求开发者明确指定输出缓冲区的大小,以避免缓冲区溢出。这有助于减少由于不安全的输出操作导致的潜在安全风险。基本语法格式如下。

```
int printf_s( size_t buffer_size, const char * format, … );
```

其中,buffer_size参数指定了输出缓冲区的总大小(以字符为单位),包括终止的空字符。编译器在运行时将检查是否可能发生缓冲区溢出,避免恶意用户触发缓冲区溢出进行攻击,从而提高了程序的鲁棒性。

拓展知识

scanf_s()和printf_s()函数是微软公司为Microsoft Visual Studio集成开发环境提供的安全标准输入输出函数,用于替代传统的scanf()和printf()函数。因此,它们主要在Windows操作系统环境下使用,特别是在支持C11标准的编译器中使用。

由于scanf_s()和printf_s()函数是微软特有的扩展,并不是C语言标准的一部分,因此在其他操作系统和编译器上,比如Linux或GCC,通常无法使用scanf_s()和printf_s()函数。这些系统和编译器通常使用标准的scanf()和printf()函数,或者提供其他的安全输入输出函数。

如果用户在非Windows操作系统平台上编程,并且需要类似scanf_s()和printf_s()函数的功能,可能需要寻找替代的安全输入输出函数,或者自己实现边界检查和缓冲区管理来确保输入输出的安全性。同时,也可以考虑使用跨平台的库或框架,这些库或框架可能提供了兼容不同平台的安全输入输出函数。

需要注意的是,即使在Windows操作系统平台上使用scanf_s()和printf_s()函数,由于它们的非标准性,代码的可移植性也可能变差。因此,在编写需要跨平台运行的代码时,建议谨慎选择输入输出函数,以确保代码在不同平台上的兼容性和安全性。

3.6　科技前沿之云计算

1. 云计算的发展历程：从“大计算机”到“云”的演变

很长一段时间以前，人们使用的大多是大型计算机，它们体积庞大、价格昂贵，只有少数机构能拥有。但随着技术的进步，人们开始想办法让计算机变得更小、更便宜，让更多人能用上。

2. 互联网的诞生

互联网的出现彻底改变了这一切。它像一张巨大的网，把全世界的计算机都连接了起来。人们可以通过网络互相交流信息，分享资源。

3. 虚拟化技术的出现

接着，虚拟化技术出现了。想象一下，你有一台计算机，你可以通过虚拟化技术，在这台计算机上再“造”出几台虚拟的计算机来。这些虚拟计算机可以独立运行不同的程序，互不干扰。

4. 云计算的诞生

后来，有人想，既然我们可以把计算机虚拟化，那为什么不把计算资源、存储资源和网络资源都放到网上，让所有人都能像用水电一样方便地使用呢？于是，云计算就诞生了。它就像一朵巨大的云，里面藏着无数的计算资源和数据，用户只需要通过网络就能访问和使用它们。

5. 云计算的前沿知识

1）混合云与多云

现在，很多企业不再只依赖一种云服务，而是选择将不同的云服务结合起来使用，这就是混合云和多云战略。比如，有的企业会把重要的数据放在自己的私有云上，而把一些不那么重要的任务交给公有云处理。

2）无服务器计算

无服务器计算听起来很神奇，但其实很简单。以前用户需要自己管理服务器，但现在可以把代码交给云服务提供商，由他们自动管理服务器，用户只需要关注自己的代码和业务逻辑就可以了。

3）容器化技术

容器化技术就像是一个个的小盒子，每个盒子里都装着一个应用。这些盒子可以轻松地从一个地方搬到另一个地方，而不用担心里面的应用会出问题。这样，应用的部署和迁移就变得非常简单和快速了。

4）AI 与云计算的结合

人工智能非常热门，而云计算为 AI 提供了强大的支持。通过云计算，AI 可以拥有更多的计算资源和数据，从而变得更加聪明和强大。

5）边缘计算

有时候，用户需要处理的数据非常多，而且要求处理速度非常快。这时候，边缘计算就

派上用场了。它可以把一些计算任务和数据存储放到离用户更近的地方,从而减少延迟并提高响应速度。

6) 量子计算与云计算

量子计算是一种全新的计算方式,它比现在的计算机要快得多。虽然现在量子计算还在发展初期,但已经有云服务提供商开始尝试将量子计算纳入他们的服务中了。未来,我们或许能通过云计算来体验量子计算的强大能力。

本章小结

本章深入探讨了C语言中的标准输入与输出函数,这是编程中不可或缺的一部分。首先学习了scanf()函数,它是用于从标准输入(如键盘)读取格式化数据的函数。通过指定不同的格式控制符,可以轻松控制输入数据的类型和格式。

紧接着,介绍了printf()函数,它是C语言中用于向标准输出(通常是终端或显示器)输出格式化字符串的强大工具。与scanf()函数类似,printf()函数也使用格式控制符来指定期望的输出类型和格式。然而,使用scanf()函数时需要特别注意缓冲区溢出的问题,因为不恰当的输入可能导致程序崩溃或产生不可预期的结果。

为了简化字符的输入输出操作,C语言还提供了putchar()和getchar()函数。putchar()函数用于向标准输出写入一个字符,而getchar()函数则从标准输入读取一个字符。这两个函数在处理字符数据时特别有用,因为它们不需要使用复杂的格式控制符。

此外,本章还介绍了输入输出函数的安全版本(scanf_s()和printf_s()),它们旨在增强输入输出的安全性,通过提供额外的参数来防止缓冲区溢出等安全问题。尽管这些函数并不是C语言标准库的一部分,但在处理敏感数据或需要更高安全性的场景中,它们是非常有用的。

本章习题

一、单选题

1. 以下程序的输出结果是(　　)。

```
int main()
{ int a = 12,b = 12;
printf(" % d  % d\n", -- a,++b);
return 0;
}
```

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

2. 有如下的语句:scanf("a=%d,b=%d,c=%d",&a,&b,&c);为使变量a的值为1,b的值为2,c的值为3,从键盘输入数据的正确形式是(　　)。

A. 32　　B. 1,3,2

C. a=1,b=2,c=3　　D. a=1,b=3,c=2

3. 运行以下程序，运行时从键盘上输入：6,5,65,66 <回车>。则输出结果是(　　)。

```
main()
{ char a,b,c,d;
scanf("%c,%c,%d,%d",&a,&b,&c,&d);
printf("c,%c,%c,%c\n",a,b,c,d);
}
```

A. 6,5,A,B　　B. 6,5,65,66　　C. 6,5,6,5　　D. 6,5,6,6

4. 以下程序的输出结果是(　　)。

```
main()
{ int a=666,b=888;
printf("%d\n",a,b);
}
```

A. 错误信息　　B. 666　　C. 888　　D. 666,888

5. 以下程序的输出结果是(　　)。

```
main()
{ int m=0256,n=256;
printf("%o %o\n",mn,n);
}
```

A. 0256 0400　　B. 0256 256　　C. 256 400　　D. 400 400

6. 以下程序的输出结果是(　　)。

```
main( )
{
int x=102, y=012;
printf("%2d,%2d\n",x,y);
}
```

A. 10,01　　B. 02,12　　C. 102,10　　D. 02,10

7. 以下程序的输出结果是(　　)。

```
main()
{ printf("%d\n",NULL); }
```

A. 0　　B. 1

C. −1　　D. NULL 没定义，出错

8. 有定义语句：int x, y;若要通过 scanf("%d,%d",&x,&y);语句使变量 x 得到数值 11，变量 y 得到数值 12，下面 4 组输入形式中，错误的是(　　)。

A. 11 12 <回车>　　B. 11,12 <回车>

C. 11,<空格>12 <回车>　　D. 11,<回车>12 <回车>

9. 以下程序的输出结果是(　　)。

```
main()
{ int a; char c=10;
float f=100.0; double x;
a=f/=c*=(x=6.5);
```

```
printf("%d %d %3.1f %3.1f\n",a,c,f,x);
}
```

A. 1 65 1 6.5　　B. 1 65 1.5 6.5

C. 1 65 1.0 6.5　　D. 2 65 1.5 6.5

10. 运行以下程序段，并从键盘上输入：10A10 <回车>，则输出结果是(　　)。

```
int m=0,n=0; char c='a';
scanf("%d%c%d",&m,&c,&n);
printf("%d,%c,%d\n",m,c,n);
```

A. 10,A,10　　B. 10,a,10　　C. 10,a,0　　D. 10,A,0

二、填空题

1. 使用语句 scanf("x=%d,y=%d",&x,&y)，输入变量 x=2，y=3，正确的输入是_____。

2. 如果要判断一个字符变量 ch 是否为小写字母，正确的判断表达式是_____。

3. 下列程序段的输出结果是_____。

```
int a=12345; printf("%4d\n", a);
```

4. 下列程序段的输出结果是_____。

```
int a=123; printf("%-02d\n", a);
```

5. 运行如下程序段，若要求 a1、a2、c1、c2 的值分别为 10、20、A、B，正确的数据输入是_____。

```
int a1,a2;
char c1,c2;
scanf("%d%c%d%c",&a1,&c1,&a2,&c2);
```

6. 有下列程序，输入数据 12345ff678，其运行结果是_____。

```
#include <stdio.h>
int main()
{
    int x;
    float y;
    scanf("%3d%f",&x,&y);
    printf("x=%d,y=%f\n",x,y);
    return 0;
}
```

7. 下列程序的运行结果是_____。

```
#include <stdio.h>
int main()
{
    float f=12.34567;
    printf("%f,%.4f,%4.3f,%10.3f ",f,f,f,f);
    return 0;
}
```

8. 下列程序的运行结果是_____。

```
#include <stdio.h>
int main()
{
    printf("%d,%c\n", '5'-'0',5+'0');
    return 0;
}
```

9. 下列程序输入 1 2 3 后的运行结果是_____。

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

10. C 语言中的_____字符是以反斜杠“\”开头，后跟规定的单个字符或数字的字符常量。

三、改错题

1. 以下程序实现的是：从键盘输入一个整数、浮点数和字符，再依次将其输出。程序有两处错误，请找出并改正。

```
int main()
{
    int n;
    float m;
    char ch;
    // 输入整数
    printf("请输入一个整数: ");
    scanf("%d", &n);
    // 输入浮点数
    printf("请输入一个浮点数: ");
    scanf("%f", &m);
    // 输入字符
    printf("请输入一个字符: ");
    scanf("%c",&ch);
    // 显示输入的内容
    printf("你输入的整数是%d\n",n);
    printf("你输入的浮点数是%.2f\n", m); // %.2f 表示保留两位小数
    printf("你输入的字符是%c\n", ch);
    return 0;
}
```

2. 以下程序实现的是：要求用户输入一个华氏温度，然后程序将其转换为摄氏温度并输出(均保留小数点后两位)。转换公式为 $C=(F-32)\times 5/9$，其中 C 代表摄氏温度，F 代表华氏温度。程序有两处错误，请找出并改正。

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

```
{
    int fahrenheit, celsius;
    // 提示用户输入华氏温度
    printf("请输入一个华氏温度: ");
    scanf("%f", fahrenheit);
    // 转换华氏温度到摄氏温度
    celsius = (fahrenheit - 32) * 5 / 9;
    // 输出摄氏温度
    printf("%.2f 华氏度等于 %.2f 摄氏度.\n", fahrenheit, celsius);
    return 0;
}
```

四、编程题(注：由于还没有学习分支结构,因此题目中所有的除法均不进行除零检查)

1. 用户输入一个 4 位数,将其反向输出。

2. 用户输入一个十进制整数,依次按十进制、八进制、十六进制(小写形式)和十六进制(大写形式)将其输出。

3. 用户依次输入一个学生的大学语文成绩、高等数学成绩、大学英语成绩,计算其平均成绩(保留整数),将其在屏幕上按照如下格式输出。

```
*********************************************
大学语文成绩：88.5
高等数学成绩：85
大学英语成绩：67.5
平均成绩：80
*********************************************
```

4. 用户由键盘输入一个 ASCII,将其对应的字符输出。

5. 用户输入出生年月日,转换形式后输出到显示器上。例如,用户输入 20030512,在显示器上输出：2003 年 5 月 12 日。

6. 用户输入字符,将其转换为对应的 ASCII 值输出。

7. 用户输入小写字母,将其转换为大写字母输出。

8. 用户输入 3 个电阻的阻值,计算其并联和串联后的电阻值(保留小数点后两位),并输出到显示器。

9. 用户分别输入分子和分母,输出其分式和对应的小数值(保留小数点后两位),示例如下。

用户输入：

分子：4

分母：5

屏幕输出：

分式：4/5

小数值：0.80

10. 用户输入两个数,分别计算它们的和、差、乘积、商,并在显示器输出(保留小数点后两位)。