一， C语言输入法的选择

计算机起源于美国，C语言、C++、Java、JavaScript 等很多流行的编程语言都是美国人发明的，所以在编写代码的时候必须使用**英文半角输入法**，尤其是标点符号，初学者一定要引起注意。

图1：搜狗输入法

一些相似的中英文标点符号：

- 中文分号；和英文分号;
- 中文逗号，和英文逗号,
- 中文冒号：和英文冒号:
- 中文括号（）和英文括号()
- 中文问号？和英文问号?
- 中文单引号‘’和英文单引号' '
- 中文双引号“ ”和英文双引号" "

全角与半角输入法的区别

全角和半角的区别主要在于除汉字以外的其它字符，比如标点符号、英文字母、阿拉伯数字等，全角字符和半角字符所占用的位置的大小不同。

在计算机屏幕上，一个汉字要占两个英文字符的位置，人们把一个英文字符所占的位置称为“半角”，相对地把一个汉字所占的位置称为“全角”。

标点符号、英文字母、阿拉伯数字等这些字符不同于汉字，在半角状态它们被作为英文字符处理，而在全角状态作为中文字符处理，请看下面的例子。

半角输入：

```
Hello C,I like!
```

全角输入：

```
Ｈｅｌｌｏ　Ｃ，Ｉ　ｌｉｋｅ！
```

另外最重要的一点是：“相同”字符在全角和半角状态下对应的编码值（例如 Unicode 编码、GBK 编码等）不一样，所以它们是不同的字符。

二，编译器的选择

我们平时所说的程序，是指双击后就可以直接运行的程序，这样的程序被称为可执行程序（Executable Program）。在 Windows 下，可执行程序的后缀有 .exe 和 .com（其中 .exe 比较常见）；在类 UNIX 系统（Linux、Mac OS 等）下，可执行程序没有特定的后缀，系统根据文件的头部信息来判断是否是可执行程序。

什么是编译器

将C语言代码转换成CPU能够识别的二进制指令，也就是将代码加工成 .exe 程序；这个工具是一个特殊的软件，叫做编译器（Compiler）。

编译器能够识别代码中的词汇、句子以及各种特定的格式，并将他们转换成计算机能够识别的二进制形式，这个过程称为编译（Compile）。

C语言的编译器有很多种，不同的平台下有不同的编译器，例如：

- Windows 下常用的是微软开发的 **cl.exe**，它被集成在 Visual Studio 或 Visual C++ 中，一般不单独使用；
- Linux 下常用的是 GUN 组织开发的 **GCC**，很多 Linux 发行版都自带 GCC；
- Mac 下常用的是 **LLVM/Clang**，它被集成在 Xcode 中（Xcode 以前集成的是 GCC，后来由于 GCC 的不配合才改为 LLVM/Clang，LLVM/Clang 的性能比 GCC 更加强大）。

什么是集成开发环境

- 编辑器：用来编写代码，并且给代码着色，以方便阅读；
- 代码提示器：输入部分代码，即可提示全部代码，加速代码的编写过程；
- 调试器：观察程序的每一个运行步骤，发现程序的逻辑错误；
- 项目管理工具：对程序涉及到的所有资源进行管理，包括源文件、图片、视频、第三方库等；
- 漂亮的界面：各种按钮、面板、菜单、窗口等控件整齐排布，操作更方便。

Visual Studio、Dev C++、Xcode、Visual C++ 6.0、C-Free、Code::Blocks 等，它们统称为集成开发环境（IDE，Integrated Development Environment）。

三，学习之前明确几个概念

1. 源文件（Source File)

保存代码的文件

每种编程语言的源文件都有特定后缀，为了方便被编译器识别，被程序员理解。

- C语言源文件的后缀是`.c`；
- C++语言（C Plus Plus）源文件的后缀是`.cpp`；
- Java 源文件的后缀是`.java`；
- Python 源文件的后缀是`.py`；
- JavaScript 源文件后置是`.js`。

源文件其实就是纯文本文件，他内部并没有特殊格式。后缀仅仅是为了表明该文件中保存的是某种语言的代码。

2. 工程/项目（project）

一个完整的程序（软件）包含多项功能，每一项功能都需要几千几万行代码。不可能都放到一个源文件当中，不仅不利于打开源文件，代码的编写和维护也变得非常困难。

将代码分门别类放到多个源文件。除了代码，一个程序往往包含图片视频，音频，控件等其他资源，他们也都是一个一个地文件。

为了有效管理这些文件，我们有理由把他们都放到一个目录（文件夹）下，并且这个目录下只存放与当前程序有关的资源。

3. 工程类型/项目类型

“程序”是一个比较宽泛的称呼，它可以细分为很多种类，例如：

- 有的程序不带界面，完全是“黑屏”的，只能输入一些字符或者命令，称为控制台程序（Console Application），例如 Windows 下的 cmd.exe，Linux 或 Mac OS 下的终端（Terminal）。
- 有的程序带界面，看起来很漂亮，能够使用鼠标点击，称为GUI程序（Graphical User Interface Program），例如 QQ、迅雷、Chrome 等。
- 有的程序不单独出现，而是作为其它程序的一个组成部分，普通用户很难接触到它们，例如静态库、动态库等。

4. 链接

编译器一次只能编译一个源文件，如果当前程序包含了多个源文件，那么就需要编译多次。编译器每次编译的结果是产生一个中间文件（可以认为是一种临时文件），而不是最终的可执行文件。中间文件已经非常接近可执行文件了，它们都是二进制格式，内部结构也非常相似。

将当前程序的所有中间文件以及系统库（暂时可以理解为系统中的一些组件）组合在一起，才能形成最终的可执行文件，这个组合的过程就叫做链接（Link）。完成链接功能的软件叫做链接器（Linker）。

如果程序只包含了一个源文件，是不是就不需要链接了呢？不是的！ 经过编译后程序虽然只有一个中间文件，不再需要和其它的中间文件组合了，但是这个唯一的中间文件还需要和系统库组合，这个过程也是链接。也就是说，不管有多少个源文件，都必须经过编译和链接两个过程才能生成可执行文件。

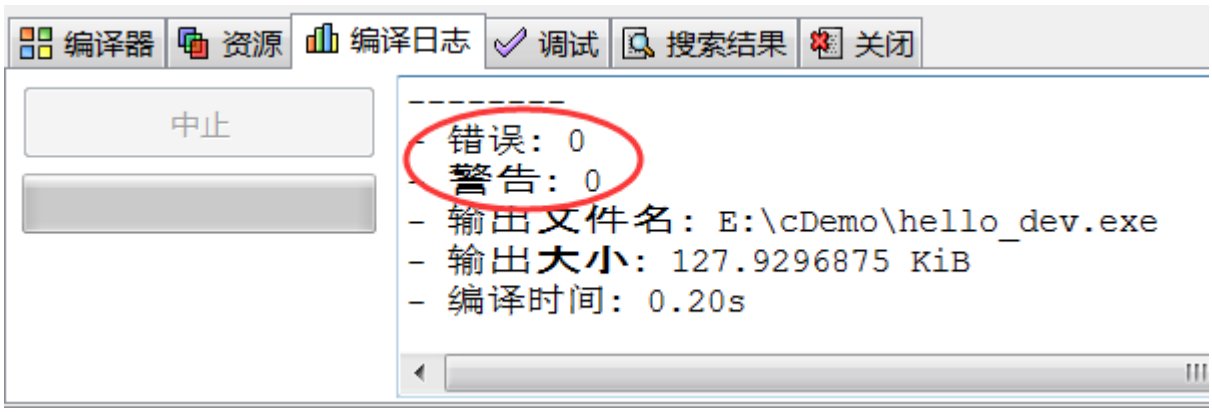
四，C语言的发展

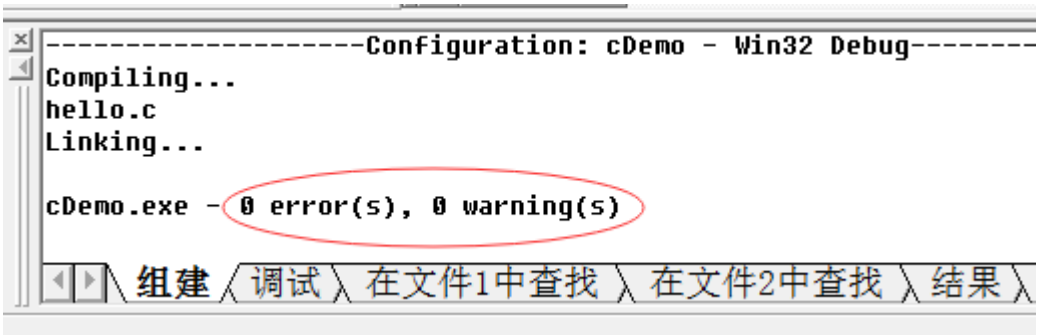
1. 1967年，26岁的丹尼斯·里奇开发完Unix之后，发现汇编语言写的Unix移植性很差，无法运行在不同型号的机器上，为了提高通用性和开发效率，丹尼斯·里奇决定发明一种新的编程语言——C语言。
2. C89标准。 到了八十年代，C语言越来越流行，各个厂商推出了多款C语言编译器，它们都根据行业和厂商自己的需求，进行了各种扩展，而C语言标准也变得层次不齐。为了统一，83年美国国家标准局（简称ANSI）成立委员会，专门制定C语言标准。1989年标准被批准，被称为ANSI X3.159-1989 "Programming Language C" . 这个版本的C语言标准通常被称为ANSI C。又由于这个版本是89年完成制定的，因此也被称为C89。
3. C90标准。与ANSI C内容基本一样，主要是格式组织不一样。ANSI C，ISO C，C89， C90.这些都是一样的。
4. C99标准。这个时候的C语言编译器基本已经成熟，各个组织对 C99 的支持所表现出来的兴趣不同。当 GCC 和其它一些商业编译器支持 C99 的大部分特性的時候，微软和 Borland 却似乎对此不感兴趣，或者说没有足够的资源和动力来改进编译器，最终导致不同的编译器在部分语法上存在差异。
5. C11标准。

C语言为什么有很多编译器。

- C语言并没有一个官方机构，也不属于哪个公司，它只有一个制定标准的委员会，任何其他组织或者个人都可以开发C语言的编译器，而这个编译器要遵守哪个C语言标准，是 100% 遵守还是部分遵守，并没有强制性的措施，也没有任何约束。换句话说，各个厂商可以为了自己的利益、根据自己的喜好来开发编译器。

五，C语言程序的错误和警告





- 错误（Error）表示程序不正确，不能正常编译、链接或运行，必须要纠正。
- 警告（Warning）表示可能会发生错误（实际上未发生）或者代码不规范，但是程序能够正常运行，有的警告可以忽略，有的要引起注意。
- 错误和警告可能发生在编译、链接、运行的任何时候。

六，分析第一个C语言程序

通过分析，对C语言有一个整体的认知

```c
#include <stdio.h>
int main()
{
    puts("Welcome to ACM!");
    return 0;
}
```

函数的概念

C语言提供了很多功能，例如输入输出、获得日期时间、文件操作等，我们只需要一句简单的代码就能够使用。

C语言的开发者们已经为我们编写了大量代码，将常见的基本功能都完成了，我们可以直接拿来使用，这些代码被分门别类的放进了不同的文件中，并且每一段代码都有唯一的名字。使用代码时，只要在对应的名字后面加上（）就可以。这样的一段代码能够独立地完成某个功能，一次编写完成后可以重复使用。可以认为，函数就是一段可以重复使用的代码。

函数的一个明显特征就是使用时必须带括号（），必要的话，括号中还可以包含待处理的数据，比如上述代码。将字符串“Welcome to ACM!”交给了puts()来处理。这就叫函数调用。

如果要处理多个数据，那么它们之间用逗号 , 分隔。

```
pow(10, 2);
```

自定义函数和main函数

C语言自带的函数称为库函数（Library Function）。库（Library）是编程中一个基本概念。C语言自带的库称为标准库。其他公司或个人开发的库称为第三方库。

除了库函数，我们还可以编写自己的函数，拓展程序的功能。自己编写的函数称为自定义函数。自定义函数与库函数再编写方式和使用方式上完全相同。

main 是函数的名字，( ) 表明这是函数定义，{ } 之间的代码是函数要实现的功能。

函数可以接收待处理的数据，同样可以将处理结果告诉我们；使用`return`可以告知处理结果。示例中第5行代码表明，main 函数的处理结果是整数 0。return 可以翻译为“返回”，所以函数的处理结果被称为返回值（Return Value）。

int 是 integer 的简写，意为“整数”。它告诉我们，函数的返回值是整数。

C语言规定，一个程序必须有且只有一个 main 函数。main 被称为主函数，是程序的入口函数，程序运行时从 main 函数开始，直到 main 函数结束（遇到 return 或者执行到函数末尾时，函数才结束）。

头文件的概念

`#include <stdio.h>`是什么意思呢

C语言开发者们编写了很多常用函数，并分门别类的放在了不同的文件，这些文件就称为头文件（header file）。每个头文件中都包含了若干个功能类似的函数，调用某个函数时，要引入对应的头文件，否则编译器找不到函数。

> 实际上，头文件往往只包含函数的说明，也就是告诉我们函数怎么用，而函数本身保存在其他文件中，在链接时才会找到。对于初学者，可以暂时理解为头文件中包含了若干函数。

#include命令仅仅是将头文件中的文本复制到当前文件，然后和当前文件一起编译。

较早的C语言标准库包含了15个头文件，stdio.h 和 stdlib.h 是最常用的两个：

- stdio 是 standard input output 的缩写，stdio.h 被称为“标准输入输出文件”，包含的函数大都和输入输出有关，puts() 就是其中之一。
- stdlib 是 standard library 的缩写，stdlib.h 被称为“标准库文件”，包含的函数比较杂乱，多是一些通用工具型函数，system() 就是其中之一。

七，C语言标识符，关键字，注释，表达式和语句

标识符

定义变量时，我们使用了诸如 a、abc、mn123 这样的名字，它们都是程序员自己起的，一般能够表达出变量的作用，这叫做标识符（Identifier）。

C语言规定，标识符只能由**字母**（A~Z, a ~ z）**数字**（0~9）和**下划线**（_）组成，并且第一个字符必须是字母或下划线，不能是数字。

关键字

关键字（Keywords）是由C语言规定的具有特定意义的字符串，通常也称为保留字，例如 int、char、long、float、unsigned 等。我们定义的标识符不能与关键字相同，否则会出现错误。

注释

注释（Comments）可以出现在代码中的任何位置，用来向用户提示或解释代码的含义。程序编译时，会忽略注释，不做任何处理，就好像它不存在一样。

C语言支持单行注释和多行注释：

- 单行注释以 `//` 开头，直到本行末尾（不能换行）；
- 多行注释以 `/*` 开头，以 `*/` 结尾，注释内容可以有一行或多行。

表达式（Expression）和语句（Statement）

表达式（Expression）和语句（Statement）的概念在C语言中并没有明确的定义：

- 表达式可以看做一个计算的公式，往往由数据、变量、运算符等组成，例如 `3*4+5`、`a=c=d` 等，表达式的结果必定是一个值；
- 语句的范围更加广泛，不一定是计算，不一定有值，可以是某个操作、某个函数、选择结构、循环等。
- 表达式必须有一个执行结果，这个结果必须是一个值，例如 `3*4+5` 的结果 17，`a=c=d=10` 的结果是 10，`printf("hello")` 的结果是 5（printf 的返回值是成功打印的字符的个数）。
- 以分号 `;` 结束的往往称为语句，而不是表达式，例如 `3*4+5;`、`a=c=d;` 等。

分号

在 C 程序中，分号是语句结束符。也就是说，每个语句必须以分号结束。它表明一个逻辑实体的结束。

例如，下面是两个不同的语句：

```
printf("Hello, World! \n");
return 0;
```

八，C语言变量，数据类型，运算符

二进制思想

二进制只有0和1两个数字，基数为2，在加减法运算中，逢二进一，借一当二。

- 表示数值：0、1、10、111、100、1000001
- 加法：1+0=1、1+1=10、10+110=1000、111+111=1110
- 减法：1-0=1、10-1=1、100-11=1、1010-101=101

内存中数据的储存

十进制数，文字，符号，图形，音频，视频，这些在计算机中在内存中都是以二进制的行书来表示。

想学好编程，就必须了解二进制

单位换算：

- 8 Bit = 1Byte
- 1024Byte = 1KB
- 1024KB = 1MB
- 1024MB = 1GB
- 1024GB = 1TB

变量

我们需要在计算机内存中找一块区域，规定它来放一些东西

比如找一块区域放一个整数

`int a;`

`int` 是 Integer 的简写，意思是整数，a 是我们给这块区域起的名字，当然也可以叫其他的，abc等等。

> 注意int 与 a 之间是有空格的。

不过`int a`仅仅是在内存中找了一块保存整数的区域。要将具体的数字放进去还需要这样操作。

`a = 123;`

`=` 是一个新符号，它在数学中是等于号，但是在C语言中，这个叫做赋值。指把数据放到内存的过程。

> int a = 123; 可以起到和上述一样的作用。

```
//a中的整数不是一成不变的，只要我们需要，随时可以更改，更改方式就是再次赋值。
int a=123;
a=1000;
a=9999;
```

因为a的值可以改变。所以我们叫它变量。

`int a;`创造了一个变量 a，我们把这个过程叫做变量定义。`a=123;`把 123 交给了变量 a，我们把这个过程叫做给变量赋值；又因为是第一次赋值，也称变量的初始化，或者赋初值。

数据类型

内存中的数据有多种解释方式，使用之前必须确定。

上面的`int a;`就表明，这份数据是整数，不能理解为其他的东西。int有一个专业的称呼，叫做数据类型。

数据类型用来说明数据的类型，确定了数据的解释方式。

说 明	字符型	短整型	整型	长整型	单精度浮点型	双精度浮点型	无类型
数据类型	char	short	int	long	float	double	void

序号	类型与描述
1	**基本类型：** 它们是算术类型，包括两种类型：整数类型和浮点类型。
2	**枚举类型：** 它们也是算术类型，被用来定义在程序中只能赋予其一定的离散整数值的变量。
3	**void 类型：** 类型说明符 *void* 表明没有可用的值。
4	**派生类型：** 它们包括：指针类型、数组类型、结构类型、共用体类型和函数类型。

连续定义多个变量

```
int a, b, c;
float m = 10.9, n = 20.56;
char p, q = '@';
```

连续定义多个变量用逗号分隔，并且要拥有相同的数据类型，变量可以初始化，也可以不初始化。

数据的长度

数据长度是指数据占用多少个字节，占用的越多，能储存的数据就越多。

下表列出了关于标准整数类型的存储大小和值范围的细节：

类型	存储大小	值范围
char	1 字节	-128 到 127 或 0 到 255
unsigned char	1 字节	0 到 255
signed char	1 字节	-128 到 127
int	2 或 4 字节	-32,768 到 32,767 或 -2,147,483,648 到 2,147,483,647
unsigned int	2 或 4 字节	0 到 65,535 或 0 到 4,294,967,295
short	2 字节	-32,768 到 32,767
unsigned short	2 字节	0 到 65,535
long	4 字节	-2,147,483,648 到 2,147,483,647
unsigned long	4 字节	0 到 4,294,967,295

下表列出了关于标准浮点类型的存储大小、值范围和精度的细节：

类型	存储大小	值范围	精度
float	4 字节	1.2E-38 到 3.4E+38	6 位小数
double	8 字节	2.3E-308 到 1.7E+308	15 位小数
long double	16 字节	3.4E-4932 到 1.1E+4932	19 位小数

运算符

- 算术运算符
  - `+ - * / % ++ --`
- 关系运算符
  - `== != > < >= <=`
- 逻辑运算符
  - && || !
- 位运算符
  - & | ^
- 赋值运算符
  - =
  - +=
  - -=
  - *=
  - /=
  - %=
  - <<=

    - `>>=`
    - &=
    - ^=
    - |=
- 杂项运算符
    - sizeof
    - & 取地址
    - `*` 解引用
    - 三元 ？：

	加法	减法	乘法	除法	求余数（取余）
数学	+	-	×	÷	无
C语言	+	-	*	/	%

对除法的说明

- 当除数和被除数都是整数时，运算结果也是整数；如果不能整除，那么就直接丢掉小数部分，只保留整数部分，这跟将小数赋值给整数类型是一个道理。
- 一旦除数和被除数中有一个是小数，那么运算结果也是小数，并且是 double 类型的小数。

```
#include <stdio.h>
int main()
{
    int a = 100;
    int b = 12;
    float c = 12.0;

    double p = a / b;
    double q = a / c;

    printf("p=%lf, q=%lf\n", p, q);

    return 0;
}
```

运行结果： p=8.000000, q=8.333333

对取余运算的说明

取余，也就是求余数，使用的运算符是 %。C语言中的取余运算只能针对整数，也就是说，% 的两边都必须是整数，不能出现小数，否则编译器会报错。

另外，余数可以是正数也可以是负数，由 % 左边的整数决定：

- 如果 % 左边是正数，那么余数也是正数；
- 如果 % 左边是负数，那么余数也是负数。

加减运算符的简写

```
#include <stdio.h>
int main()
{
    int a = 12;
    int b = 10;

    printf("a=%d\n", a);

    a = a + 8;
    printf("a=%d\n", a);

    a = a * b;
    printf("a=%d\n", a);

    return 0;
}
```

`a = a # b`可以简写为：`a #= b`

表示 +、-、*、/、% 中的任何一种运算符。

```
int a = 10, b = 20;
a += 10;  //相当于 a = a + 10;
a *= (b-10);  //相当于 a = a * (b-10);
a -= (a+20);  //相当于 a = a - (a+20);
```

C语言自增（++）和自减（--）

一个整数类型的变量自身加 1 可以这样写：

```
a = a + 1;
a += 1;
a++;
++a;
```

相应的，也有`a--`和`--a`，它们叫做自减，表示自身减 1。

`++`和`--`分别称为自增运算符和自减运算符，它们在循环结构中使用很频繁。

需要重点说明的是，++ 在变量前面和后面是有区别的：

- ++ 在前面叫做前自增（例如 ++a）。前自增先进行自增运算，再进行其他操作。
- ++ 在后面叫做后自增（例如 a++）。后自增先进行其他操作，再进行自增运算。

```c
#include <stdio.h>
int main()
{
    int a = 10, b = 20, c = 30, d = 40;
    int a1 = ++a, b1 = b++, c1 = --c, d1 = d--;

    printf("a=%d, a1=%d\n", a, a1);
    printf("b=%d, b1=%d\n", b, b1);
    printf("c=%d, c1=%d\n", c, c1);
    printf("d=%d, d1=%d\n", d, d1);

    return 0;
}
```

输出结果：

a=11, a1=11

b=21, b1=20

c=29, c1=29

d=39, d1=40

运算符优先级

优先级	运算符	名称或含义	使用形式	结合方向	说明
1	[]	数组下标	数组名[常量表达式]	左到右	
()	圆括号	（表达式）/函数名(形参表)			
.	成员选择（对象）	对象.成员名			
->	成员选择（指针）	对象指针->成员名			
2	-	负号运算符	-表达式	右到左	单目运算符
(类型)	强制类型转换	(数据类型)表达式			
++	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		
*	取值运算符	*指针变量	单目运算符		
&	取地址运算符	&变量名	单目运算符		
!	逻辑非运算符	!表达式	单目运算符		
~	按位取反运算符	~表达式	单目运算符		
sizeof	长度运算符	sizeof(表达式)			
3	/	除	表达式/表达式	左到右	双目运算符
*	乘	表达式*表达式	双目运算符		
%	余数（取模）	整型表达式/整型表达式	双目运算符		
4	+	加	表达式+表达式	左到右	双目运算符
-	减	表达式-表达式	双目运算符		
5	<<	左移	变量<<表达式	左到右	双目运算符
>>	右移	变量>>表达式	双目运算符		
6	>	大于	表达式>表达式	左到右	双目运算符
>=	大于等于	表达式>=表达式	双目运算符		
<	小于	表达式<表达式	双目运算符		
<=	小于等于	表达式<=表达式	双目运算符		

优先级	运算符	名称或含义	使用形式	结合方向	说明
7	==	等于	表达式==表达式	左到右	双目运算符
!=	不等于	表达式!= 表达式	双目运算符		
8	&	按位与	表达式&表达式	左到右	双目运算符
9	^	按位异或	表达式^表达式	左到右	双目运算符
10	|	按位或	表达式|表达式	左到右	双目运算符
11	&&	逻辑与	表达式&&表达式	左到右	双目运算符
12	||	逻辑或	表达式||表达式	左到右	双目运算符
13	?:	条件运算符	表达式1? 表达式2: 表达式3	右到左	三目运算符
14	=	赋值运算符	变量=表达式	右到左	
/=	除后赋值	变量/=表达式			
*=	乘后赋值	变量*=表达式			
%=	取模后赋值	变量%=表达式			
+=	加后赋值	变量+=表达式			
-=	减后赋值	变量-=表达式			
<<=	左移后赋值	变量<<=表达式			
>>=	右移后赋值	变量>>=表达式			
&=	按位与后赋值	变量&=表达式			
^=	按位异或后赋值	变量^=表达式			
|=	按位或后赋值	变量|=表达式			
15	,	逗号运算符	表达式,表达式,...	左到右	从左向右顺序运算

九，C语言的输入输出

scanf()函数

scanf 是 scan format 的缩写，意思是格式化扫描，也就是从键盘获得用户输入

```c
#include <stdio.h>
int main()
{
    int a = 0, b = 0, c = 0, d = 0;
    scanf("%d", &a);  //输入整数并赋值给变量a
    scanf("%d", &b);  //输入整数并赋值给变量b
    printf("a+b=%d\n", a+b);  //计算a+b的值并输出
    scanf("%d %d", &c, &d);  //输入两个整数并分别赋值给c、d
    printf("c*d=%d\n", c*d);  //计算c*d的值并输出
    return 0;
}
```

scanf() 有两个以空格分隔的`%d`，后面还跟着两个变量，这要求我们一次性输入两个整数，并分别赋值给 c 和 d。注意`"%d %d"`之间是有空格的，所以输入数据时也要有空格。对于 scanf()，输入数据的格式要和控制字符串的格式保持一致。

scanf() 格式控制符汇总

格式控制符	说明
%c	读取一个单一的字符
%hd、%d、%ld	读取一个十进制整数，并分别赋值给 short、int、long 类型
%ho、%o、%lo	读取一个八进制整数（可带前缀也可不带），并分别赋值给 short、int、long 类型
%hx、%x、%lx	读取一个十六进制整数（可带前缀也可不带），并分别赋值给 short、int、long 类型
%hu、%u、%lu	读取一个无符号整数，并分别赋值给 unsigned short、unsigned int、unsigned long 类型
%f、%lf	读取一个十进制形式的小数，并分别赋值给 float、double 类型
%e、%le	读取一个指数形式的小数，并分别赋值给 float、double 类型
%g、%lg	既可以读取一个十进制形式的小数，也可以读取一个指数形式的小数，并分别赋值给 float、double 类型
%s	读取一个字符串（以空白符为结束）

scanf()细节

C语言中有多个函数可以从键盘获得用户输入

- scanf()：与printf类似，可以处理多种类型的数据
- getchar(),getche(),getch()：都用于输入单个字符
- gets()：获取一行数据，并作为字符串处理

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

```c
int main()
{
    int a, b, c;

    scanf("%d %d", &a, &b);
    printf("a+b=%d\n", a+b);

    scanf("%d   %d", &a, &b);
    printf("a+b=%d\n", a+b);

    scanf("%d, %d, %d", &a, &b, &c);
    printf("a+b+c=%d\n", a+b+c);

    scanf("%d is bigger than %d", &a, &b);
    printf("a-b=%d\n", a-b);

    return 0;
}
```

```c
#include <stdio.h>
int main()
{
    int a = 1, b = 2, c = 3, d = 4;  //修改处：给变量赋予不同的初始值
    scanf("%d", &a);
    scanf("%d", &b);
    printf("a=%d, b=%d\n", a, b);
    scanf("%d %d", &c, &d);
    printf("c=%d, d=%d\n", c, d);

    return 0;
}
```

运行结果：

```
12 60 a10↙
a=12, b=60
c=3, d=4
```

前两个整数被正确读取后，剩下了 a10，而第三个 scanf() 要求输入两个十进制的整数，a10 无论如何也不符合要求，所以只能读取失败。输出结果也证明了这一点，c 和 d 的值并没有被改变。

getchar()

getchar()，它就是`scanf("%c", c)`的替代品

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

getche()

getche() 就比较有意思了，它没有缓冲区，输入一个字符后会立即读取，不用等待用户按下回车键，这是它和 scanf()、getchar() 的最大区别。请看下面的代码：

```c
#include <stdio.h>
#include <conio.h>
int main()
{
    char c = getche();
    printf("c: %c\n", c);
    return 0;
}
```

getch()

getch() 也没有缓冲区，输入一个字符后会立即读取，不用按下回车键，这一点和 getche() 相同。getch() 的特别之处是它没有回显，看不到输入的字符。所谓回显，就是在控制台上显示出用户输入的字符；没有回显，就不会显示用户输入的字符，就好像根本没有输入一样。

回显在大部分情况下是有必要的，它能够与用户及时交互，让用户清楚地看到自己输入的内容。但在某些特殊情况下，我们却不希望有回显，例如输入密码，有回显是非常危险的，容易被偷窥。

```c
#include <stdio.h>
#include <conio.h>
int main()
{
    char c = getch();
    printf("c: %c\n", c);
    return 0;
}
```

对三个函数的总结

函数	缓冲区	头文件	回显	适用平台
getchar()	有	stdio.h	有	Windows、Linux、Mac OS 等所有平台
getche()	无	conio.h	有	Windows
getch()	无	conio.h	无	Windows

输入字符串

输入字符串当然可以使用 scanf() 这个通用的输入函数，对应的格式控制符为`%s.` gets() 这个专用的字符串输入函数，它拥有一个 scanf() 不具备的特性。

- scanf() 读取字符串时以空格为分隔，遇到空格就认为当前字符串结束了，所以无法读取含有空格的字符串。
- gets() 认为空格也是字符串的一部分，只有遇到回车键时才认为字符串输入结束，所以，不管输入了多少个空格，只要不按下回车键，对 gets() 来说就是一个完整的字符串。

printf()函数

格式控制符	说明
%c	输出一个单一的字符
%hd、%d、%ld	以十进制、有符号的形式输出 short、int、long 类型的整数
%hu、%u、%lu	以十进制、无符号的形式输出 short、int、long 类型的整数
%ho、%o、%lo	以八进制、不带前缀、无符号的形式输出 short、int、long 类型的整数
%#ho、%#o、%#lo	以八进制、带前缀、无符号的形式输出 short、int、long 类型的整数
%hx、%x、%lx %hX、%X、%lX	以十六进制、不带前缀、无符号的形式输出 short、int、long 类型的整数。如果 x 小写，那么输出的十六进制数字也小写；如果 X 大写，那么输出的十六进制数字也大写。
%#hx、%#x、%#lx %#hX、%#X、%#lX	以十六进制、带前缀、无符号的形式输出 short、int、long 类型的整数。如果 x 小写，那么输出的十六进制数字和前缀都小写；如果 X 大写，那么输出的十六进制数字和前缀都大写。
%f、%lf	以十进制的形式输出 float、double 类型的小数
%e、%le %E、%lE	以指数的形式输出 float、double 类型的小数。如果 e 小写，那么输出结果中的 e 也小写；如果 E 大写，那么输出结果中的 E 也大写。
%g、%lg %G、%lG	以十进制和指数中较短的形式输出 float、double 类型的小数，并且小数部分的最后不会添加多余的 0。如果 g 小写，那么当以指数形式输出时 e 也小写；如果 G 大写，那么当以指数形式输出时 E 也大写。
%s	输出一个字符串

printf()细节以及进阶

C语言有三个函数可以用来在显示器上输出数据

- puts()：只能输出字符串
- putchar()：只能输出单个字符
- printf(): 可以输出各种类型的数据。

细节用法

```
#include <stdio.h>
int main()
{
    int a1=20, a2=345, a3=700, a4=22;
    int b1=56720, b2=9999, b3=20098, b4=2;
    int c1=233, c2=205, c3=1, c4=6666;
    int d1=34, d2=0, d3=23, d4=23006783;
    printf("%d        %d       %d       %d\n", a1, a2, a3, a4);
    printf("%d     %d      %d     %d\n", b1, b2, b3, b4);
    printf("%d       %d       %d         %d\n", c1, c2, c3, c4);
    printf("%d        %d         %d        %d\n", d1, d2, d3, d4);
    return 0;
}
```

```
20        345       700       22
56720     9999      20098     2
233       205       1         6666
34        0         23        23006783
```

```
#include <stdio.h>
int main()
{
    int a1=20, a2=345, a3=700, a4=22;
    int b1=56720, b2=9999, b3=20098, b4=2;
    int c1=233, c2=205, c3=1, c4=6666;
    int d1=34, d2=0, d3=23, d4=23006783;
    printf("%-9d %-9d %-9d %-9d\n", a1, a2, a3, a4);
    printf("%-9d %-9d %-9d %-9d\n", b1, b2, b3, b4);
    printf("%-9d %-9d %-9d %-9d\n", c1, c2, c3, c4);
    printf("%-9d %-9d %-9d %-9d\n", d1, d2, d3, d4);
    return 0;
}
```

```
20        345       700       22
56720     9999      20098     2
233       205       1         6666
34        0         23        23006783
```

printf() 格式控制符的完整形式如下：

```
%[flag][width][.precision]type
```

> [ ] 表示此处的内容可有可无，是可以省略的。

1. type表示输出类型，比如%d，%f，%c，%lf。
2. width表示最小输出宽度，也就是至少占用几个字符的位置
    1. 例如，`%-9d`中 width 对应 9，表示输出结果最少占用 9 个字符的宽度。
    2. 当输出结果的宽度不足 width 时，以空格补齐（如果没有指定对齐方式，默认会在左边补齐空格）；当输出结果的宽度超过 width 时，width 不再起作用，按照数据本身的宽度来输出。
3. **.precision** 表示精度，也就是小数的位数
4. 当小数部分的位数大于 precision 时，会按照四舍五入的原则丢掉多余的数字；
5. 当小数部分的位数小于 precision 时，会在后面补 0。
6. 另外，.precision 也可以用于整数和字符串，但是功能却是相反的：
    - 用于整数时，.precision 表示最小输出宽度。与 width 不同的是，整数的宽度不足时会在左边补 0，而不是补空格。
    - 用于字符串时，.precision 表示最大输出宽度，或者说截取字符串。当字符串的长度大于 precision 时，会截掉多余的字符；当字符串的长度小于 precision 时，.precision 就不再起作用。

```
#include <stdio.h>
int main(){
    int n = 123456;
    double f = 882.923672;
    char *str = "abcdefghi";
    printf("n: %.9d  %.4d\n", n, n);
    printf("f: %.2lf  %.4lf  %.10lf\n", f, f, f);
    printf("str: %.5s  %.15s\n", str, str);
    return 0;
}
```

```
n: 000123456  123456
f: 882.92  882.9237  882.9236720000
str: abcde  abcdefghi
```

4. flag是标志字符

标志字符	含义
-	`-`表示左对齐。如果没有，就按照默认的对齐方式，默认一般为右对齐。
+	用于整数或者小数，表示输出符号（正负号）。如果没有，那么只有负数才会输出符号。
空格	用于整数或者小数，输出值为正时冠以空格，为负时冠以负号。
#	对于八进制（%o）和十六进制（%x / %X）整数，# 表示在输出时添加前缀；八进制的前缀是 0，十六进制的前缀是 0x / 0X。对于小数（%f / %e / %g），# 表示强迫输出小数点。如果没有小数部分，默认是不输出小数点的，加上 # 以后，即使没有小数部分也会带上小数点。

```
#include <stdio.h>
int main(){
    int m = 192, n = -943;
    float f = 84.342;
    printf("m=%10d, m=%-10d\n", m, m);  //演示 - 的用法
    printf("m=%+d, n=%+d\n", m, n);  //演示 + 的用法
    printf("m=% d, n=% d\n", m, n);  //演示空格的用法
    printf("f=%.0f, f=%#.0f\n", f, f);  //演示#的用法
    return 0;
}
```

十，资源推荐

1. 慕课C语言
2. C语言中文网
3. 中国大学慕课
4. CSDN，博客园

十一，OJ(Online Judge)的使用方法

Online Judge系统（简称OJ）是一个在线的判题系统。用户可以在线提交程序多种程序（如C、C++、Pascal）源代码，系统对源代码进行编译和执行，并通过预先设计的测试数据来检验程序源代码的正确性。

简单来讲：就是把你写的代码提交给oj，然后oj经过数据评测，会返回给你几个结果，列举如下：

- **Pending** ：系统忙，你的答案在排队等待.
- **Pending Rejudge**: 因为数据更新或其他原因，系统将重新判你的答案.
- **Compiling** : 正在编译.
- **Running & Judging**: 正在运行和判断.
- **Accepted** : 程序通过!

- **Presentation Error** : 答案基本正确，但是格式不对。
- **Wrong Answer** : 答案不对，仅仅通过样例数据的测试并不一定是正确答案，一定还有你没想到的地方.
- **Time Limit Exceeded** : 运行超出时间限制，检查下是否有死循环，或者应该有更快的计算方法。
- **Memory Limit Exceeded** : 超出内存限制，数据可能需要压缩，检查内存是否有泄露。
- **Output Limit Exceeded**: 输出超过限制，你的输出比正确答案长了两倍.
- **Runtime Error** : 运行时错误，非法的内存访问，数组越界，指针漂移，调用禁用的系统函数。请点击后获得详细输出。
- **Compile Error** : 编译错误，请点击后获得编译器的详细输出。

而只有AC(Accepted)，这道题才算完全通过。凡是出现PE，WA，TLE，MLE，RE，CE，都要做出调整，否则此题不算通过。