

C 语言教程讲义

Potato

2024 年 6 月 21 日

1 标准输出和变量

1.1 printf 函数

打开 visual studio，在“源文件”上右键，点击“添加”->“新建项”，创建一个叫 main.c 的文件，在其中输入

```
1  #include <stdio.h>
2
3  int main() {
4      printf("Hello World!");
5
6      return 0;
7  }
```

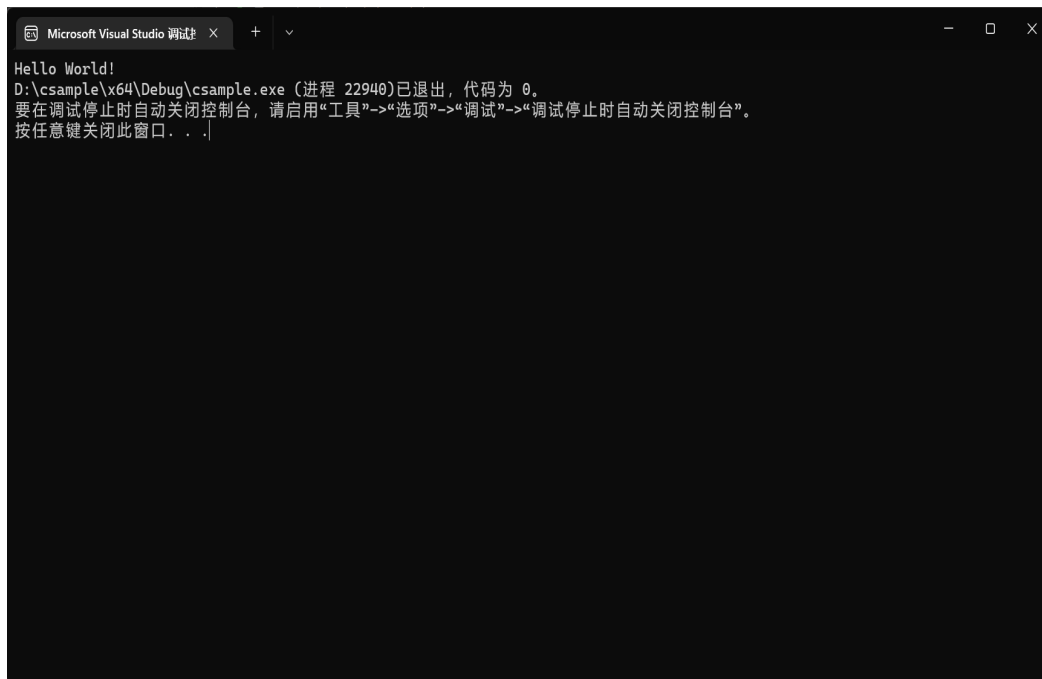
点击“本地 Windows 调试器”，将会出现类似于这样的界面

恭喜你写出了你的第一段代码。计算机领域有一个传统：刚入门编程的人的第一段代码一定要是 Hello World，这是我们对无垠的计算机世界打的招呼。这个黑乎乎的界面叫做终端（这里不做关于终端、shell 等的详细区分），我们以后会常常和它打交道。

我们发现，我们写在 printf 的引号内的内容会被打印出来（顾名思义，printf 就是 print function 嘛），因此我们可以多写几个 printf（不要忘了每行最后的英文分号），如

```
1  #include <stdio.h>
2
3  int main() {
4      printf("Hello World!");
5      printf("Potato");
6
7      return 0;
8  }
```

运行代码，我们观察到 Hello World 和 Potato 输出在了同一行。我们想要的是这二者各自占一行，那就需要在 Hello World!后加一个换行符，也即 \n，代码就变成了



```
1  #include <stdio.h>
2
3  int main() {
4      printf("Hello World!\n");
5      printf("Potato\n");
6
7      return 0;
8  }
```

这次的结果就是我们想要的了。为什么 Potato 也要加换行符？这是因为也许我们在其后还会添加新的 printf，要是到时候忘了给 Potato 加换行符，不就没有实现我们想要的效果吗？为了避免以后出现不必要的问题，我们推荐每个 printf 后都加换行符。

由上面的代码，我们总结出了这些要点：首先，我们要想输出信息，就要将其写在 printf 的引号内；其次，要想换行，就需要在输出的文本最后加一个换行符 \n；另外我们还注意到，我们先输出了 Hello World，再输出了 Potato，也就是说，程序是按照从上到下的顺序执行的。

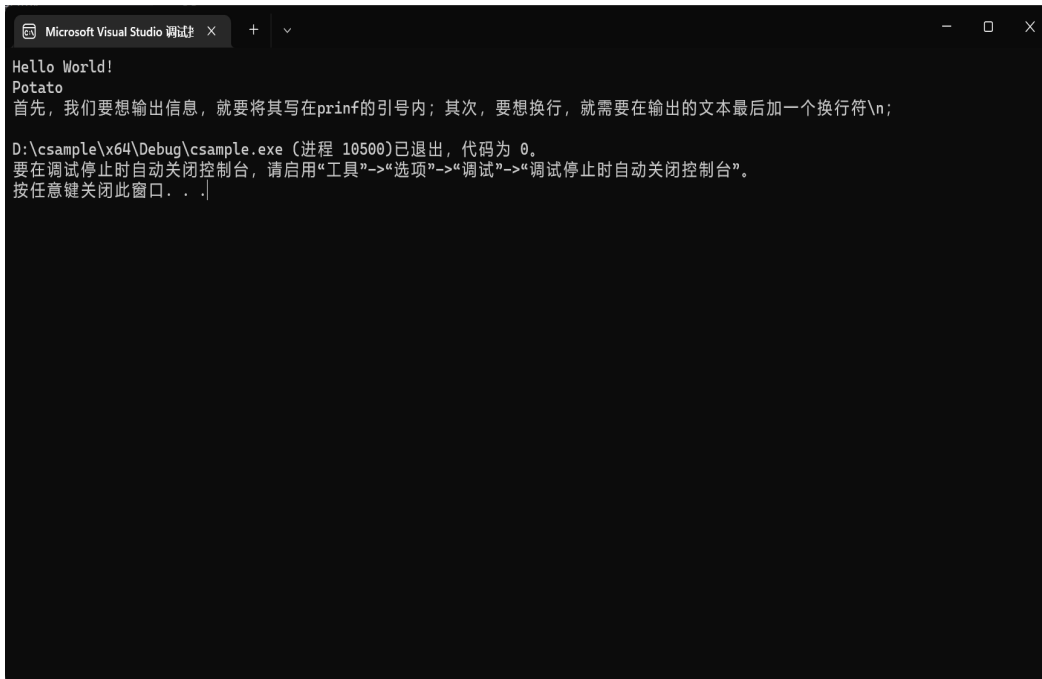
这是很重要的结论，所以我们使用 printf 输出一遍加强印象吧。于是我们写下了：

```
1  printf("首先,我们要想输出信息,就要将其写在printf的引号内;其次,要想换行,就需要在输出的文本最后加一个换行符\n;\n");
```

我们注意到有个问题，就是第一个换行符是我们想要按照文本形式输出的，但是如果按照上面的写法，它会被用于换行而不是直接输出。为了解决这个问题，我们把代码改成这样即可：

```
1  printf("首先,我们要想输出信息,就要将其写在printf的引号内;其次,要想换行,就需要在输出的文本最后加一个换行符\\n;\n");
```

运行结果是：



```
Microsoft Visual Studio 调试
Hello World!
Potato
首先，我们要想输出信息，就要将其写在printf的引号内；其次，要想换行，就需要在输出的文本最后加一个换行符\n;
D:\csample\x64\Debug\csample.exe (进程 10500)已退出，代码为 0。
要在调试停止时自动关闭控制台，请启用“工具”->“选项”->“调试”->“调试停止时自动关闭控制台”。
按任意键关闭此窗口。 . . |
```

为什么会这样呢？实际上，我们输入的 `\n` 和 `\\` 都叫做转义符号，用来实现一些特殊的输出效果。它的语法就是反斜杠 (`\`) 加一个字母。我们最常用的转义符号就是换行符 `\n`，实际上还有 `\t`、`\a` 等等。因为反斜杠会和下一个字母结合变成转义符号，所以要想输出反斜杠，就需要按照上面的例子那样使用两个反斜杠，这个转义符号的结果就是输出一个反斜杠。

1.2 变量

变量顾名思义，就是可以变的量。一个变量肯定要有名字，然后要有一个值。这就像是我们的考试成绩，有一个叫做“数学成绩”的变量，这次可以是 120，下次可以是 130。在 c 语言中，我们这样声明变量：`type name`，啥意思呢，比如我们要声明数学成绩 (`math_grade`)，那么就可以这样写

```
1 int math_grade;
```

这里的 `int` 是整数的意思，还有其它类型（如小数）我们之后会讲到。所以这行代码的意思就是，我们声明了一个叫做 `math_grade` 的变量，它是一个整数。接下来就要给它赋值了，在大部分计算机语言中，赋值号都是 `=`，比如数学成绩是 120，那么就可以写

```
1 math_grade = 120;
```

这段代码的意思就是给 `math_grade` 赋予 120 这个值。也即，我们将赋值号右边的值给了左边的变量。我们强调是将右边的值赋给左边，说明了这不是数学中的等号，它不意味着左右两边相等，也不能交换左右两边的位置。

现在我们的 `main.c` 应该是这样的

```

1  #include <stdio.h>
2
3  int main() {
4      int math_grade;
5      math_grade = 120;
6
7      return 0;
8  }

```

第四行是声明变量，第五行是给变量赋值。如果我们在声明变量时就知道它要赋什么值，那么就可以把声明和赋值写在一起

```

1  int math_grade = 120;

```

我们可以这样打印整数变量

```

1  #include <stdio.h>
2
3  int main() {
4      int math_grade;
5      math_grade = 120;
6
7      printf("%d\n", math_grade);
8
9      return 0;
10 }

```

运行程序，输出了 120。我们目前只需要这样理解：引号内的 %d 是一个占位符，引号后面的 math_grade 就是用来替换占位符的。因此我们就可以实现一些更复杂的输出了，比如

```

1  printf("数学成绩为:%d\n", math_grade);

```

现在拓展一下，如果我们定义了数学、语文和英语三门课的成绩，也即

```

1  int math_grade = 120;
2  int chinese_grade = 100;
3  int english_grade = 140;

```

如何输出呢？我们只需要写三个占位符即可，也就是

```

1  printf("数学成绩为:%d, 语文成绩为:%d, 英语成绩为:%d\n", math_grade, chinese_grade, english_grade);

```

想象一下，假如我们定义了很多变量，有数学期中成绩，数学期末成绩，语文小测成绩，语文期末成绩……我们在代码里看这么多东西就头大，所以我们希望能用自然语言给它们加上注释，便于我们理解代码。在 c 语言中，有两种形式的注释：单行注释和多行注释。

单行注释是两个斜杠 (//)，这两个斜杠之后就可以随便写字了，比如

```

1  #include <stdio.h>
2
3  int main() {
4      int math_grade; // 声明数学成绩
5      math_grade = 120; // 给数学成绩赋值120
6
7      printf("%d\n", math_grade); // 打印数学成绩
8
9      return 0;
10 }

```

多行注释的语法是/* */，在这之间可以随便写字，比如

```

1  #include <stdio.h>
2
3  int main() {
4      /*
5       该程序的展示了变量的声明与赋值语法，
6       以及使用printf打印整型变量的方法
7      */
8      int math_grade; // 声明数学成绩
9      math_grade = 120; // 给数学成绩赋值120
10
11     printf("%d\n", math_grade); // 打印数学成绩
12
13     return 0;
14 }

```

有了变量，肯定也要有常量。在 c 语言中，我们使用#define 来声明常量。比如，圆周率 π 就是一个常量，我们可以这样在 c 语言中定义圆周率

```

1  #define PI 3

```

也即，定义常量的语法是#define 常量名 常量的值。

我们定义圆周率是 3，是一个整数，因此可以按照上文的方法将其打印出来

```

1  #include <stdio.h>
2
3  #define PI 3
4
5  int main() {
6      printf("圆周率的近似值是%d\n", PI);
7
8      return 0;
9  }

```

你可能觉得奇怪，为什么定义常量的语法和变量区别这么大呢？既不需要写赋值号，也不需要标明类型。实际上，`#define` 是一个宏，在 c 语言中，程序被编译运行前，会有一个叫做预处理器的程序将所有宏替换为对应的值，也就是说（在只考虑 `#define` 宏的情况下），上面的代码经过预处理器会变成这样

```
1  #include <stdio.h>
2
3  int main() {
4      printf("圆周率的近似值是%d\n", 3);
5
6      return 0;
7  }
```

发现了吗？我们定义的常量 `PI` 被直接替换成了它的值 `3`，既然是直接替换，就相当于写死在了程序里，没法修改，所以就是常量了。实际上，我们行首的 `#include` 也是一个宏，它的效果和 `#define` 类似。关于宏和预处理的知识我们在之后会学习到。

1.3 scanf 函数

`scanf` 函数是 `scan function`，它用来从终端获取输入的文本。这个东西涉及到一些我们还没有学到的知识，但是我们后续的学习很难离开它，因此需要在这里对其简单介绍，我们目前不需要了解它的原理，会用就可以。

首先，打开 `visual studio`，在 `main.c` 中输入一段这样的代码

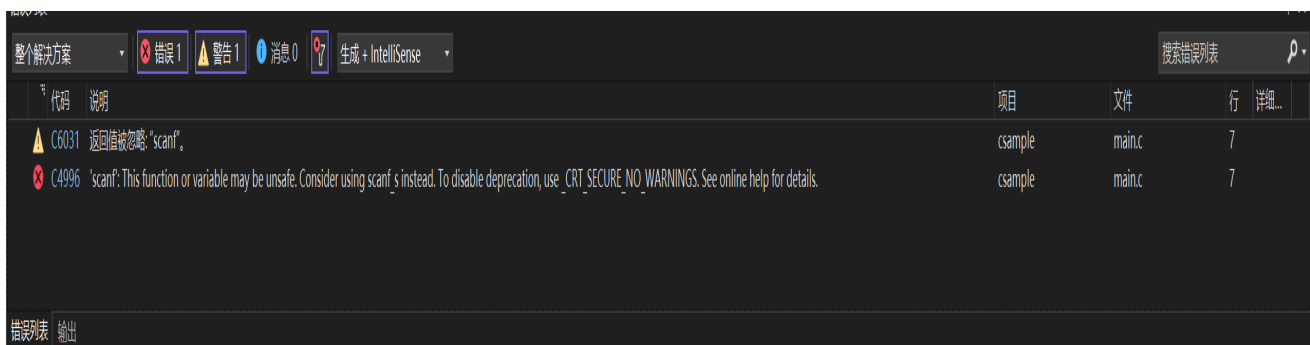
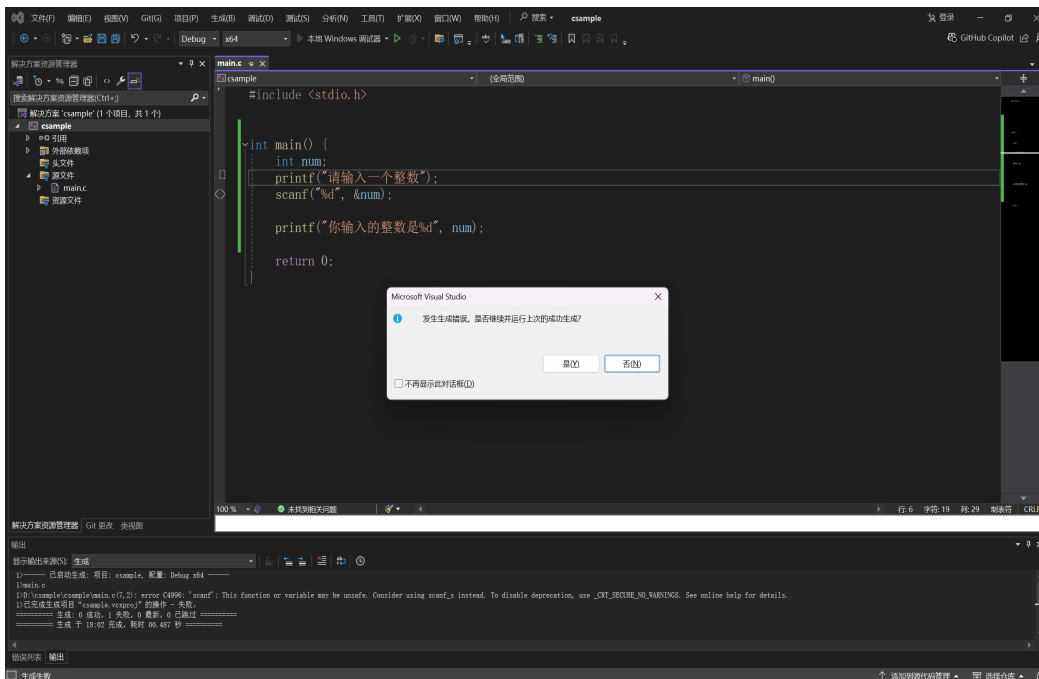
```
1  #include <stdio.h>
2
3  int main() {
4      int num; // 用于储存用户输入的数字
5      printf("请输入一个整数"); // 提示用户进行输入
6      scanf("%d", &num); // 将输入的数字保存在变量num中,注意不要忘了num前的&
7
8      printf("你输入的整数是%d", num);
9
10     return 0;
11 }
```

点击运行，出现了意料之外的情况

这个窗口的意思是我们的代码有问题，过不了编译。在实际编程开发时，由于我们很难做到一次就尽善尽美，所以常常会遇到这个窗口。还好代码的试错成本是很低的，我们只需要着手去修补自己的代码即可。我们点击否，注意到 `visual studio` 最下面出现了这样的文本

带有黄色警示标志的叫做警告（`warning`），它的意义是代码可能有一些不符合规范的地方，需要修改，但是不改也能正常运行。而红色标志的叫做错误（`error`），它的意义是这里的代码完全有问题，过不了编译，程序运行不了。我们必须消除代码中所有的 `error`，也要尽可能消除所有的 `warning`。

我们开始着手消灭这个 `error`，它的提示信息说 “This function or variable may be unsafe. Consider using `scanf_s` instead. To disable deprecation, use `_CRT_SECURE_NO_WARNINGS`. See online help



for details.”

翻译成中文，就是“这个函数或者变量可能不安全。考虑使用 scanf_s 替代。若要取消废弃（检查），使用 _CRT_SECURE_NO_WARNINGS。详细信息见在线帮助。”。它的意思就是说，我们用的这个 scanf 函数不安全，已经废弃了，要么我们换用 scanf_s，要么我们加上 _CRT_SECURE_NO_WARNINGS 来强行使用废弃的函数。

实际上，scanf 由于设计不当，会有缓冲区溢出（buffer overflow）的风险，所以我们确实不应该使用它。但是呢，只有我们使用的 visual studio 会报这个错误，其它 ide 不会强制要求你换，而且大部分的 c 语言试题和 c 语言教程仍在使用 scanf，所以我们采取强制使用废弃函数的方法来消除这个 error。

在 visual studio 的项目右键，点击属性，点击“C/C++”，点击“预处理器”，点击“预处理器定义”，随后点击右边出现的小箭头，点击“<编辑>”，单独一行输入“_CRT_SECURE_NO_WARNINGS”，点击“确定”即可。

此时我们再运行代码，终端首先变成这样