



C 语言课程讲义

作者：LMXZ

时间：2025-10-21

版本：0.1

目录

第一章 快速入门	1
1.1 快速配置开发工具	1
1.2 第一个程序，Hello World!	1
1.3 输入输出	1
1.4 分支选择、数据类型、算术运算	2
1.5 Online Judge 练习平台	3
1.6 学习方法	3
第一章 练习	3
第二章 循环、数组、函数	5
2.1 循环	5
2.1.1 for 循环	5
2.1.2 while 循环	5
2.1.3 do-while 循环	5
2.2 数组	6
2.2.1 数组声明	6
2.2.2 数组访问	6
2.3 函数	7
2.3.1 前置知识：表达式	7
2.3.2 函数的定义和调用	7
2.4 课程前瞻	8
第二章 练习	8

第一章 快速入门

1.1 快速配置开发工具

前期建议使用易配置的 Code::Blocks 作为编程工具。可以在这里下载：

<https://www.codeblocks.org/downloads/binaries/>

选择“Windows(64 bit)”（如果你用的是 Windows 的话）下的 codeblocks-X.Xmingw-setup.exe。这个版本自带了 GCC 编译器，安装后即可使用。


使用 Code::Blocks 只是为了快速上手，但并不推荐使用作为长期的开发工具，等你熟悉了 C 语言之后，可以考虑使用更强大的 IDE，比如 Visual Studio Code、CLion 等。

1.2 第一个程序，Hello World!

Hello World 是指输出“Hello, World”的简单程序，在开始学习一门语言时，一般会从这里开始入手。

```
#include <stdio.h>

int main() {
    printf("Hello, World!"); // 输出 Hello World
    return 0; // 程序结束
}
```

现在程序写好了，需要运行时就点击上面的“”按钮。

目前你需要了解的是：

- `#include <stdio.h>` 是预处理指令，用于包含标准输入输出库，提供 `printf` 函数的定义；
- `int main()` 是主函数的定义，程序从这里开始执行；
- `printf("Hello, World!");` 用于输出内容到屏幕，括号内的内容是要输出的字符串；
- `return 0;` 表示程序正常结束，返回值为 0；
- “// 输出 Hello World”这种字样是注释，用于做批注，以便人类能看懂；计算机并不会理会“//”以及后面的内容；
- 在 `int main` 里面的每一行的结尾都必须有一个分号“;”；
- 输入程序的时候要特别注意，把输入法关掉，否则你输入的符号会变成全角字符；程序中的符号应该统一为半角字符。

1.3 输入输出

刚才的例子中，`printf("XX")` 是用于输出内容的。然而一个程序往往需要输入、计算、输出三步。我们看第二个例子：

```
#include <stdio.h>

int main() {
    int a, b; // 定义变量 a b;
    printf("Please input the value of a and b:");
    scanf("%d%d", &a, &b); // 读入 a b
    int c = a + b; // 计算 a+b 的结果，存到变量 c 里面
}
```

```
printf("The result of %d + %d is: %d", a, b, c); // 输出结果
return 0;
}
```

这个程序会读取用户输入的两个数字，相加并输出。运行程序，输入“3 5”（注意中间有空格），你会看到程序输出了“The result of 3 + 5 is: 8”。

这里你需要记住三句话的用法：

- `int` 用于定义整数型变量，一行可以定义多个变量；定义时如果在后面加上“= xx”，就会同时给它赋值。
- `scanf` 用于输入，有两个 `%d`，这个东西被称为“占位符”；要读取几个变量，引号中就要有多少个 `%d`；
- `printf` 用于输出变量，其中 `%d` 也是占位符，每个占位符会被依次替换为后面的每个表达式的值。

1.4 分支选择、数据类型、算术运算

```
#include <stdio.h>
#include <math.h> // 引入数学库，使用 sqrt 函数

int main() {
    double a, b, c; // 定义三个双精度浮点数变量
    printf("Solve the equation ax^2+bx+c=0\n");
    printf("Please input the coefficients a, b and c:");
    scanf("%lf%lf%lf", &a, &b, &c); // 读入 a b c
    double delta = b * b - 4 * a * c; // 计算判别式
    if (delta < 0) {
        printf("No real roots.\n");
    } else if (delta == 0) {
        double root = -b / (2 * a);
        printf("One real root: %lf\n", root);
    } else {
        double root1 = (-b + sqrt(delta)) / (2 * a);
        double root2 = (-b - sqrt(delta)) / (2 * a);
        printf("Two real roots: %lf and %lf\n", root1, root2);
    }
    return 0;
}
```

这个例子展示了如何使用分支选择语句 `if-else` 来处理不同的情况。相比于前面的例子，这里你需要了解的内容有：

- `double` 用于定义双精度浮点数变量，可以存储小数，与 `int` 不同；
- 在 `scanf` 和 `printf` 中，使用 `%lf` 来表示双精度浮点数的占位符；
- 使用算术运算符“+”、“-”、“*”、“/”、“%”进行加减乘除和取模（即除法求余数）运算；
- 使用数学库中的函数，比如这里的 `sqrt()` 用于计算平方根；使用这些函数前需要包含相应的头文件：`#include <math.h>`；其中还有很多常用的数学函数，如 `pow()`（幂运算）、`sin()`、`cos()`（三角函数）等；
- 使用 `if-else` 语句根据条件执行不同的代码块。

1.5 Online Judge 练习平台

Online Judge (OJ) 是指在线编程练习平台, 提供了大量的编程题目供练习。常见的 OJ 平台有: 洛谷、Codeforces、牛客网、LeetCode 等。

简单介绍一下洛谷平台的使用方法:

- 访问洛谷网站: <https://www.luogu.com.cn/>, 注册并登录;
- 浏览题库, 选择适合自己水平的题目进行练习;
- 阅读题目描述, 理解题意和输入输出要求;
- 在本地编写代码, 调试通过后, 将代码提交到洛谷平台;
- 提交后, 系统会自动评测代码的正确性和效率, 并给出结果反馈: AC 表示正确, 其他结果表示有错误或超时等问题;
- 选择提交时要选择 C 语言作为编程语言;
- 严格遵守输入输出格式, 不要输出多余内容; 比如 1.3 节中的程序就不适合直接提交到 OJ 平台, 因为它包含了提示信息, 去除 “Please input...” 和 “The result...”, 只输出一个数字表示结果即可。

1.6 学习方法

这里能讲到的内容毕竟有限, 学习过程中需要善于查找资料, 遇到问题时可以:

- 查阅 C 语言相关书籍和在线文档, 如 **C Reference**; 这是最靠谱, 最权威的参考资料;
- 利用搜索引擎查找相关问题的解决方案;
- 查阅编程社区和论坛, 向其他程序员请教问题;
- AI 助手 (比如 ChatGPT) 也是一个不错的选择, 可以用来解答编程相关的问题。

擅于自主学习和解决问题是成为优秀程序员的重要能力。

练习

1. 在洛谷通过以下题目: **P1001 A+B Problem**、**P1150 Peter 的烟**、**P1425 小鱼的游泳时间**。
2. 现在你已经掌握了基本的输入、计算、输出三步; 接下来请你编写一个程序, 读取平面上三个点的坐标, 计算并输出它们构成的三角形的三条边的长度、三个角的角度以及三角形面积。

参考答案:

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

int main() {
    const double PI = acos(-1.0);
    double x1, y1, x2, y2, x3, y3;
    printf("Please input the coordinates of three points (x1 y1 x2 y2 x3 y3):");
    scanf("%lf%lf%lf%lf%lf%lf", &x1, &y1, &x2, &y2, &x3, &y3);

    double a = sqrt((x2 - x3) * (x2 - x3) + (y2 - y3) * (y2 - y3));
    double b = sqrt((x1 - x3) * (x1 - x3) + (y1 - y3) * (y1 - y3));
    double c = sqrt((x1 - x2) * (x1 - x2) + (y1 - y2) * (y1 - y2));

    double angleA = acos((b*b + c*c - a*a) / (2*b*c)) * 180.0 / PI;
    double angleB = acos((a*a + c*c - b*b) / (2*a*c)) * 180.0 / PI;
    double angleC = 180.0 - angleA - angleB;

    double s = (a + b + c) / 2;
    double area = sqrt(s * (s - a) * (s - b) * (s - c));

    printf("Sides: a=%.2lf, b=%.2lf, c=%.2lf\n", a, b, c);
    printf("Angles: A=%.2lf°, B=%.2lf°, C=%.2lf°\n", angleA, angleB, angleC);
    printf("Area: %.2lf\n", area);

    return 0;
}
```

第二章 循环、数组、函数

2.1 循环

2.1.1 for 循环

for 循环是最常用的循环结构，特别适合在已知循环次数的情况下使用。

```
for (int i = 1; i <= 5; i++) {  
    printf("%d_", i);  
}
```

语法说明：

int i = 1: 循环变量初始化

i <= 5: 循环条件

i++: 循环变量更新

在较老版本的 C（如 C89/ANSI C）标准下，不能在 for 语句头中声明循环变量，需要在循环外先声明变量，例如：

```
int i, j;  
for (i = 1; i <= 5; i++) {  
    for (j = 1; j <= 5; j++)  
        printf("%d,_%d\n", i, j);  
}
```

注意：这样声明的 i 在循环结束后仍然可见（作用域为所在块），如需限制作用域可另起一个内层块。

2.1.2 while 循环

while 循环在不确定循环次数，但知道循环条件时使用。

```
int count = 1;  
// 同样打印数字1到5  
while (count <= 5) {  
    printf("%d_", count);  
    count++;  
}
```

2.1.3 do-while 循环

do-while 循环与 while 循环类似，但会先执行一次循环体，再检查条件。

```
int count = 1; // 把 count 改为 6 或以上的值，循环体代码也至少会执行一次  
do {  
    printf("%d_", num);  
    num++;  
} while (num <= 5);
```

2.2 数组

2.2.1 数组声明

在不同的 C 标准中，数组的定义方式有区别。在 C99 及更老的版本中，数组大小只能是常量，而之后的版本中可以为变量。

```
const int BUF_LEN = 16;
int fixed_buf1[10]; // 常量长度
int fixed_buf2[BUF_LEN]; // 符号常量也是可以的
// 以下定义方式在老版本是不可用的
int len1 = 16, len2 = 0;
scanf("%d", &len2);
int buf1[len1], buf2[len2]; // 无论是自己赋值的变量还是用户输入的变量，都可以
```

动态分配是通用的数组定义方式：

```
#include <stdlib.h>

int main {
    int n = 0;
    scanf("%d", &n);
    int *p = malloc(n * sizeof(int));
    if (p == NULL) {
        // 内存分配失败的处理
    }
    for (int i = 0; i < n; ++i)
        scanf("%d", &p[i])
    // 一顿操作以后，如果数组不再使用了，记得释放内存！！
    free(p);
    return 0;
}
```

兼容性提示：若需要同时支持可能不含可变长度数组的实现，可以用预处理判断并退回到 malloc 方案：

```
/* 如果实现不支持 VLA，则 __STDC_NO_VLA__ 会被定义 */
#ifdef __STDC_NO_VLA__
    int arr[n];
#else
    int *arr = malloc(n * sizeof *arr);
    /* 使用后记得 free(arr); */
#endif
```

好复杂是不是？**现在先记住第一个示例就行了**。作为平时的练习，你在做 OJ 上的题目时一般不会使用动态分配的方式，但在一些实际工程中动态分配是到处都有的。

2.2.2 数组访问

数组下标从 0 开始，到 $n-1$ 结束（ n 为数组长度），**不包含 n** 。当你访问了超过这个范围的下标，就会发生越界。如果你想要让数组下标从 1 开始，可以把数组长度开得比实际使用的大小大一点。和现代的各种语言不同，默认情况下，程序**不会**自动帮你检查数组下标是否越界并报错，当你越界时，会出现很多匪夷所思的 bug。

那么有没有不默认的情况？有的！这一点在绝大多数国内外教材中都不会提及，但我们后面会介绍。

2.3 函数

2.3.1 前置知识：表达式

C 程序中，几乎一切操作和运算都是通过表达式实现的。举个例子：

```
int a = 0;
int b = a++;
int c = ++a;
int d = scanf("%d%d%d", &a, &b, &c);
```

其中 0、a++、++a 都是表达式。一个表达式包含以下要素：

- 对现有变量、输入输出进行的操作（也可以不做操作）；
- 返回值（可以为空）。

以 a++ 举例，它进行的操作就是将变量 a 的值加一，返回值是加一前的变量的值。

scanf("%d%d%d", &a, &b, &c) 也是表达式，它的操作是读取三个变量，返回值是读取到的变量的数量。

表达式可以通过运算符、函数等组合成新的表达式：

```
int e = (scanf("%d%d%d", &a, &b, &c) + ++a * b--) * c++ / --d ^ abs(a + b);
```

2.3.2 函数的定义和调用

函数的定义包括以下要素：

- 参数列表；
- 函数体，即进行的操作；
- 返回值类型。

可以看出函数和表达式是很像的。

调用时需要传递参数，有返回值的话也可以使用返回值做进一步处理。

举个例子：

```
#include <stdio.h>

int global_arr[10], u = 0;

float func(int x, float y, int arr[]) {
    for (int i = 0; i < 10; i++) {
        scanf("%d", &arr[i]); // 函数中不仅可以做运算，还可以输入输出
    }
    float res = ((float)x) / 10 + x / 7 + y;
    for (int i = 0; i < 10; i++) {
        res += ++arr[i]; // 这里的操作会影响到数组中的元素
    }
    x++; // 操作后的 x 会变化，但不会改变 main 中 a 的值
    u++;
    printf("x, u after change: %d, %d\n", x, u);
    return res; // 函数返回类型不是 void 的话，就必须有返回值
}

int main() {
```

```

int a;
float b;
scanf("%d%f", &a, &b);
float result = func(a, b, global_arr) + a; // 调用，并接收返回值做进一步运算
printf("Result: %.2f\n", result);
for (int i = 0; i < 10; i++) {
    printf("global_arr[%d]=%d\n", i, global_arr[i]); // 输出会显示数组里的元素都加了 1
}
print("a_after_func: %d\n", a); // a 不会因为 x++ 而变化
print("u_after_func: %d\n", u); // 但 u 会变化
return 0;
}

```

- 传递的参数在函数内部和外部被看作是不同的变量，内部改变并不会影响到函数外，但对全局变量的操作是有效的；
- 数组也可以作为参数传递，但和单一变量参数不同，函数中数组内元素的改变会影响到函数外；可以这么理解：传递的数组包含的是数组的位置信息，而不是数组内部元素的值。

2.4 课程前瞻

目前我是这么想的，现在基础知识差不多了，对于写大多数简单的题目已经没问题了。接下来一段时间先带你刷刷题，把基础打好。我们从这个题单开始：**CF 新手 100 题**。每周你就在这个题单里面随便找几个题做一下，具体多少数量我现在也不清楚，因为刷题期间实际讲课节奏我心里也没谱，等刷起来以后我们才知道。

要刷到什么程度？大概就是你遇到这个难度的题都能做出来就可以了。

由于前面我只关注你能快速上手实践，很多基础内容都忽略了，刷题期间我会顺带把这部分补上。这些内容也很重要，之所以前面不讲是因为这些你在大多数教材上都能看到，而且对于“能快速上手实践”其实关系不是很大。然后还会讲一些绝大多数国内外课程都不会教的东西。

刷题阶段完成后，接下来就开始一些简单的算法和数据结构，你不去打 ICPC 之类的竞赛的话，不用学的那么变态，掌握基础就行了。

练习

1. 为程序 2.3.2 随机构造几组输入，先手动计算输出，然后运行程序看一下和你的预期是否一致；如果不一致，看看你哪里理解错了。