Re：从零开始的C语言

一、基本数据类型

C语言中，数据有不同的类型，就像生活中我们把东西分为“整数”、“小数”、“文字”一样。编译器需要知道数据的类型才能正确存储和运算。

类型	关键字	说明	常见例子
整型	`int`	整数（通常占4字节）	`int age = 18;`
短整型	`short`	范围较小的整数	`short count = 100;`
长整型	`long`	范围较大的整数	`long distance = 123456L;`
单精度浮点型	`float`	小数（6-7位有效数字）	`float pi = 3.14f;`
双精度浮点型	`double`	更精确的小数（15-16位有效数字）	`double e = 2.71828;`
字符型	`char`	单个字符（用单引号）	`char grade = 'A';`

补充说明：

- `void` 类型表示“无类型”，通常用于函数返回值（表示没有返回值）。
- `_Bool` 类型（C99）表示布尔值（0或1），一般用 `<stdbool.h>` 中的 `bool`。

示例：定义变量并输出

```c
#include <stdio.h>

int main() {
    int a = 10;
    float b = 3.14;
    char c = 'x';
    printf("a=%d, b=%f, c=%c\n", a, b, c);
    return 0;
}
```

二、基本语法

基本语法就是写代码的“规矩”，就像写作文要遵守标点符号一样。

1. 语句与分号

- 每个执行语句后面必须以 **分号(;)** 结尾。

```c
int x = 5;   // 正确
x = x + 1;   // 正确
```

2. 注释

注释是写给程序员看的，编译器会忽略。

- 单行注释：`// 这是注释`
- 多行注释：`/* 注释内容 */`

3. 变量命名规则

- 只能由字母、数字、下划线组成，且不能以数字开头。
- 不能使用C语言关键字（如 `int`，`if`，`return` 等）。
- 区分大小写（`Age` 和 `age` 不同）。

4. 输入与输出

- 输出：`printf("格式控制", 变量列表)`
- 输入：`scanf("格式控制", &变量列表)`

```c
#include <stdio.h>
int main() {
    int age;
    printf("请输入你的年龄: ");
    scanf("%d", &age);          // & 取地址符，后面指针部分会讲
    printf("你的年龄是 %d 岁\n", age);
    return 0;
}
```

5. 运算符简介

- 算术运算符：`+ - * / %`（% 是求余数）
- 赋值运算符：`=`，以及复合赋值 `+=`，`-=` 等
- 关系运算符：`> < >= <= == !=`（注意 `==` 是相等比较）
- 逻辑运算符：`&&`（与），`||`（或），`!`（非）

三、基本程序结构

一个C程序无论长短，都遵循相似的骨架结构。就像一栋房子有地基、墙和屋顶。

```c
// 1. 头文件包含（告诉编译器要用到的功能）
#include <stdio.h>

// 2. 函数声明（可选）
void sayHello();
```

```c
// 3. 主函数 -- 程序的入口，每个程序必须有且仅有一个 main 函数
int main() {
    // 4. 变量定义
    int num = 0;

    // 5. 执行语句
    printf("Hello, World!\n");
    sayHello();

    // 6. 返回 0 表示程序正常结束
    return 0;
}

// 7. 其他函数定义
void sayHello() {
    printf("这是自定义函数\n");
}
```

常见结构类型：

顺序结构

代码从上往下一条一条执行。

选择结构（分支）

```c
if (条件) {
    // 条件为真时执行
} else {
    // 条件为假时执行
}
```

或者 `switch-case` 多分支。

循环结构

- `while` 循环：先判断后执行
- `do-while` 循环：先执行一次，再判断
- `for` 循环：最常用，结构清晰

```
// for 循环示例：打印 1~10
for (int i = 1; i <= 10; i++) {
    printf("%d ", i);
}
```

四、自定义函数

函数就是把一段代码包装起来，给它起个名字。需要的时候“调用”这个名字，而不用重复写代码。这就像我们把常用工具放在工具箱里。

1. 函数的定义格式

```
返回类型 函数名(参数列表) {
    // 函数体
    return 返回值;  // 如果返回类型为 void，则不需要 return
}
```

2. 例子：无参数无返回值

```c
#include <stdio.h>

void showMenu() {
    printf("===== 菜单 =====\n");
    printf("1. 开始游戏\n");
    printf("2. 退出\n");
}

int main() {
    showMenu();   // 调用函数
    return 0;
}
```

3. 例子：带参数和返回值

```c
// 计算两个整数的和
int sum(int a, int b) {
    int result = a + b;
    return result;
}

int main() {
    int x = 3, y = 5;
    int s = sum(x, y);
    printf("和 = %d\n", s);  // 输出 8
    return 0;
}
```

4. 形参与实参

- **形式参数**：定义函数时括号里的变量（如 `int a, int b`）
- **实际参数**：调用函数时传入的具体值（如 `sum(3, 5)` 中的 3 和 5）

5. 函数声明

如果函数的定义写在 `main` 后面，需要先在前面“声明”一下，告诉编译器有这个函数存在。

```
#include <stdio.h>
// 函数声明
int max(int x, int y);

int main() {
    printf("%d\n", max(10, 20));
    return 0;
}

// 函数定义
int max(int x, int y) {
    return (x > y) ? x : y;
}
```

五、指针

指针是C语言的精华，也是初学者觉得最难的部分。请放轻松，我们用“房间和门牌号”来理解。

- 每个变量都存放在内存的某个位置，这个位置有一个**地址**（就像酒店的房间号）。
- **指针**就是存储这个地址的变量（就像一张写着房号的卡片）。

1. 指针的声明和取地址

- `&` 运算符：取得变量的地址。
- `*` 运算符：声明指针变量时表示“这是一个指针”；用在变量前表示取指针指向的内容（解引用）。

```
int age = 18;      // 普通整型变量
int *p;            // 声明一个指向 int 类型的指针变量 p
p = &age;          // p 保存了 age 的地址

printf("age 的值：%d\n", age);   // 18
printf("age 的地址：%p\n", &age); // 比如 0x7ffc...
printf("p 中保存的地址：%p\n", p); // 和上面地址相同
printf("p 指向的内容：%d\n", *p);  // 18
```

2. 通过指针修改变量的值

```
int num = 10;
int *ptr = &num;
*ptr = 20;         // 通过指针修改 num 的值
printf("num = %d\n", num);  // 输出 20
```

3. 指针与函数参数（传地址调用）

C语言的函数参数默认是“传值”的（复制一份给函数）。如果想在函数内部修改外部的变量，需要传递指针。

```
#include <stdio.h>

void swap(int *a, int *b) {
    int temp = *a;
```

```c
    *a = *b;
    *b = temp;
}

int main() {
    int x = 3, y = 5;
    swap(&x, &y);   // 传递地址
    printf("x=%d, y=%d\n", x, y);  // 输出 x=5, y=3
    return 0;
}
```

4. 指针与数组

数组名本质上就是数组第一个元素的地址（可以理解为指向数组首元素的指针）。

c

```c
int arr[3] = {10, 20, 30};
int *p = arr;        // 等价于 int *p = &arr[0];

printf("%d\n", *p);      // 输出 10
printf("%d\n", *(p+1));  // 输出 20（指针移动一个 int 大小）
```

5. 空指针

- 一个指针可以赋值为 `NULL`，表示它不指向任何地方。
- 解引用 `NULL` 指针会导致程序崩溃，所以要小心。

```c
int *ptr = NULL;
if (ptr != NULL) {
    *ptr = 100;   // 安全地使用
}
```

6. 指针的小结

概念	符号	例子	含义
取地址	`&`	`&x`	获得变量 x 的地址
声明指针	`*`	`int *p;`	p 是一个指向 int 的指针
解引用	`*`	`*p`	访问 p 所指向的变量

> 记住：指针就是地址，地址就是内存中的位置编号。把指针变量想象成一把钥匙，拿着它能打开对应的房间，修改里面的东西。

附：综合示例（把所学内容串起来）

下面这个程序演示了数据类型、输入输出、函数、指针的配合使用。

```c
#include <stdio.h>

// 自定义函数：通过指针修改数值
void addFive(int *n) {
    *n = *n + 5;
}

int main() {
    // 基本数据类型
    int score = 0;
    float average = 0.0;
```

```
    char grade = '?';

    // 输入
    printf("请输入你的分数（整数）：");
    scanf("%d", &score);

    // 使用指针
    int *pScore = &score;
    printf("通过指针看到的分数：%d\n", *pScore);

    // 调用函数修改变量
    addFive(&score);
    printf("加5分后：%d\n", score);

    // 简单判断等级
    if (score >= 90) grade = 'A';
    else if (score >= 70) grade = 'B';
    else if (score >= 60) grade = 'C';
    else grade = 'D';

    average = score * 0.9;  // 随便算个平均值玩
    printf("等级：%c，加权平均：%.2f\n", grade, average);

    return 0;
}
```