函数

函数的定义

我们知道，如果程序的逻辑比较复杂、代码量比较大，或者是重复性的功能比较多，那么全部写在主函数里就会显得十分冗长和杂乱。为了使代码更加简洁、思路更加清晰，C语言提供了“函数”。函数是一个实现一定功能的语句的集合，并在需要的时候可以反复调用而不必每次都重新写一遍。像 `math.h` 头文件下面的 `sin()`、`pow()` 等数学函数就是系统已经帮实现好其功能的、我们可以直接使用的函数。那么，如果我们需要自定义函数的内容的话，应该如何做？下面我们就来讲解一下函数的定义和使用方法。

先给出基本语法：

```
返回类型 函数名称(参数类型 参数) {
    函数主体
}
```

举一个例子来说明：

```
#include <stdio.h>
void print1() {
    printf("Haha,\n");
    printf("Good idea!\n");
}
void print2() {
    printf("Ohno,\n");
    printf("Bad idea!\n");
}
int main() {
    print1();
    print2();
    return 0;
}
```

输出结果：

```
Haha,
Good idea!
Ohno,
Bad idea!
```

可以看到，`print1()` 和 `print2()` 就是两个自定义的函数，分别都实现了输出两个语句的功能。我们把 `print1()` 函数提出来分析一下：

```
void print1() {
    printf("Haha,\n");
    printf("Good idea!\n");
}
```

对比前面给出的函数基本格式，很容易知道 `print1` 就是函数名称，大括号内部的两个 `printf` 语句就是函数实体、也就是 `print1` 函数需要实现的功能。

接下来我们可以看到，对应于“返回类型”的地方我们写的是 `void`。`void` 的含义是“空”，即不返回任何东西，如果我们的自定义函数只是单纯实现一些语句而不返回变量，那么这里就可以填写 `void`，表示返回类型为空。

最后，在 `print1` 后面的小括号里没有填写任何参数，我们把这种不需要提供参数就可以执行的函数称为无参函数，而把 `fabs(x)`、`pow(r, p)` 这种需要填写参数的函数称为有参函数。

下面我们来看一个有参函数的例子：

```
#include <stdio.h>
int judge(int x) {
    if(x > 0) return 1;
    else if(x == 0) return 0;
    else return -1;
}
int main() {
    int a, ans;
    scanf("%d", &a);
    ans = judge(a);
    printf("%d\n", ans);
    return 0;
}
```

输入一个整数：

```
-4
```

输出结果：

```
-1
```

这份代码中，`judge()` 函数就是有参函数。同时我们还发现，`judge()` 函数有了返回类型 ——`int` 型（是否是有参函数跟是否有返回类型无关）。这说明在这个函数的运行过程中需要返回一个 `int` 型的常量或变量。C 语言中使用 `return` 来返回函数需要传回的数据，且 `return` 后面的数据类型要和一开始给出的返回类型相同。

再来看 `judge()` 函数的参数。可以知道 `judge()` 函数需要从外部传入一个 `int` 型的变量 `x`，然后判断 `x` 的正负号：如果 `x` 是负数，则返回 `-1`（一个 `int` 型常量）；如果 `x` 是 `0`，则返回 `0`；如果 `x` 是正数，则返回 `1`。而在主函数这边，有一句为 `ans = judge(a)`，这里将 `a` 作为 `judge` 的参数传入，然后将返回的 `int` 型数据赋值给 `ans`。

细心的学弟妹可能会发现，`judge` 函数的参数写的是 `int x`，但是下面传入的参数却是 `a`。变量名不同真的可以吗？这需要介绍两个概念——**全局变量**和**局部变量**。

（1）全局变量

全局变量是指在定义之后的所有程序段内都有效的变量，例如下面这个例子：

```
#include <stdio.h>
int x;
void change() {
    x = x + 1;
}
int main() {
    x = 10;
    change();
    printf("%d\n", x);
    return 0;
}
```

输出结果：

```
11
```

在这份代码中，我们把 `x` 定义在所有函数的前面，这样在 `x` 定义之后的所有程序段都共用这个 `x`，所以当主函数对 `x` 赋值为 `10` 之后，使用 `change()` 函数可以改变 `x` 的值，从而令 `x` 变为 `11`。

（2）局部变量

与全局变量相对，局部变量定义在函数内部，且只在函数内部生效，函数结束时局部变量销毁。例如下面这个例子：

```
#include <stdio.h>
void change(int x) {
    x = x + 1;
}
int main() {
    int x = 10;
    change(x);
    printf("%d\n", x);
    return 0;
}
```

输出结果：

```
10
```

我们看到，当我们在主函数中定义了 `x` 之后，将其作为 `change()` 函数的参数传入，并令 `x` 加 `1`，但是最后输出的时候 `x` 却仍然是 `10`。这是因为 `change` 函数的参数 `x` 为局部变量，仅在函数内部生效，通过 `change(x)` 传进去的 `x` 其实只是传进去一个副本，也即 `change` 函数的参数 `x` 和 `main` 函数里的 `x` 其实是作用于两个不同函数的不同变量（虽然名字相同）。这种传递参数的方式叫做值传递，函数定义的小括号内的参数称为形式参数或形参，而把实际调用时候小括号内的参数称为实际参数或实参。因此，如果想要让定义的变量在所有函数都有用，最好还是使用全局变量的定义方式。

最后指出，函数的参数个数可以不止一个，多于一个的情况只需要用逗号隔开，传入参数的时候位置对应即可。例如下面这个例子：

```
#include <stdio.h>
int MAX(int a, int b, int c) {
    int M;
    if(a >= b && a >= c)
        M = a;
    else if(b >= a && b >= c)
        M = b;
    else M = c;
    return M;
}
int main() {
    int a, b, c;
    scanf("%d%d%d", &a, &b, &c);
    printf("%d\n", MAX(a, b, c));
    return 0;
}
```

输入三个整数：

```
3 5 4
```

输出结果：

```
5
```

这份代码实现了输入三个整数，然后输出三个整数中的最大值。

再谈 main 函数

主函数对一个程序来说只能有一个，并且无论主函数写在哪个位置，整个程序一定是从主函数的第一个语句开始执行，然后在需要调用其它函数的时候才去调用。在本篇一开始就介绍了主函数，但是没有对其写法进行解释。现在我们来看看 `main` 函数是个什么结构：

```
int main() {
    ......
    return 0;
}
```

现在我们以函数的眼光来看它：`main` 是函数名称；小括号内没有填写东西，因此是无参函数（可以有参数，但不需要了解）；返回类型是 `int` 型，并且在函数主体的最后面 `return` 了 `0`。对计算机来说，`main` 函数返回 `0` 的意义在于告知系统程序正常终止。

以数组作为函数参数

函数的参数也可以是数组，且数组作为参数时，参数中数组的第一维不需要填写长度（如果是二维数组，那么第二维需要填写长度），实际调用的时候也只需要填写数组名。最重要的是，数组作为参数时，在函数中对数组元素的修改就等同于是对原数组元素的修改（这与普通的局部变量不同）。正如下面这个例子：

```
#include <stdio.h>
void change(int a[], int b[][5]) {
    a[0] = 1;
    a[1] = 3;
    a[2] = 5;
    b[0][0] = 1;
}
int main() {
    int a[3] = {0};
    int b[5][5] = {0};
    change(a, b);
    for(int i = 0; i < 3; i++) {
        printf("%d\n", a[i]);
    }
    return 0;
}
```

输出结果：

```
1
3
5
```

不过，虽然数组可以作为参数，但是却不允许作为返回类型出现。如果想要返回数组，只能用上面的方法，将想要返回的数组作为参数传入。

函数的嵌套调用

函数的嵌套调用是指在一个函数中调用另一个函数，调用方式和之前 `main` 函数调用其它函数是一样的。例如下面这个程序：

```
#include <stdio.h>
int max_2(int a, int b) {
    if(a > b) return a;
    else return b;
}
int max_3(int a, int b, int c) {
    int temp = max_2(a, b);
    temp = max_2(temp, c);
    return temp;
}
int main(){
    int a, b, c;
    scanf("%d%d%d", &a, &b, &c);
    printf("%d\n", max_3(a, b, c));
    return 0;
}
```

输入三个整数：

```
3 5 4
```

输出结果：

```
5
```

上面的代码可以求解三个整数中的最大值，`main` 函数先调用 `max_3` 函数，在 `max_3` 中又调用了 `max_2` 函数来比较两个整数的大小。

函数的递归调用（这也是901考试的重点）

函数的递归调用是指一个函数调用该函数自身。这是一个重要的概念，不过我们把它放到数据结构部分去讲述，此处学弟妹只需要知道递归是函数自己调用自己的过程，类似于下面的代码计算了 `n` 的阶乘。

```c
#include <stdio.h>
int F(int n) {
    if(n == 0) return 1;
    else return F(n - 1) * n;
}
int main() {
    int n;
    scanf("%d", &n);
    printf("%d\n", F(n));
    return 0;
}
```

输入数据：

```
3
```

输出结果：

```
6
```