第3章 chapter 3

函　数

本章学习目标

- 熟练掌握函数的定义与声明
- 理解有参函数与无参函数
- 理解形式参数与实际参数
- 理解作用域
- 熟练掌握 printf 函数与 scanf 函数
- 熟练掌握 putchar 函数与 getchar 函数

如果程序的功能比较多，规模比较大，把所有的程序代码都写在一个主函数中，就会使主函数变量庞杂、头绪不清，使阅读和维护程序变得困难。C 语言提供了可采用“组装”的办法来简化程序设计的过程。例如，组装一台计算机，事先生产好各种部件（如电源、主板、硬盘驱动器、风扇等），在最后组装计算机时，用到什么就从仓库里取出什么，直接装上就可以了，这就是模块化程序设计的思路，每个模块可以是一个或多个函数。

3.1　函数的定义与声明

3.1.1　函数的定义

函数也称为方法，是指实现某种功能的代码块，例如，要实现输出一行文字的功能，可自定义一个函数来实现，示例代码如下：

```
void remember()
{
    printf("拼搏到无能为力,坚持到感动自己!");
}
```

void 表示该函数没有返回值，remember 是为函数取的名字，remember 后面有一对小括号，小括号中代表函数的参数，假如没有参数，小括号内为空。函数的主体从左大括号开始，到右大括号结束，中间是函数的功能，该函数实现输出“拼搏到无能为力，坚持到感动自己!”。

注意：

有些老式的编译器不能识别 void，在这些编译器中需要将 void 改为 int，int 表示函数要返回一个整数，因此还要象征性地为函数增加一个返回语句：return 0。

remember 这个函数不是 C 函数库中的函数，而是自己写的，也叫自定义函数，它的名字是 remember，如要用到这个函数的时候就可以写：

```
remember();
```

这就是调用函数，程序执行到这里，就会立即跳转到 remember 函数的定义部分去执行（定义部分就是实现函数功能的部分），当函数执行完毕后，再跳回到原始位置继续往下执行。这个过程就好比在学习疲惫时，突然想起老师对自己说的话，心中又充满了力量，接下来通过一个案例来演示函数的定义及调用，具体如例 3-1 所示。

例 3-1

```
1  #include <stdio.h>
2  void remember()
3  {
4      printf("拼搏到无能为力，坚持到感动自己！\n");
5  }
6  int main()
7  {
8      printf("疲惫的时候，总有个声音在呼喊！\n");
9      remember();
10     printf("想起这段话，心中又充满了力量！\n");
11     return 0;
12 }
```

输出：

```
疲惫的时候，总有个声音在呼喊！
拼搏到无能为力，坚持到感动自己！
想起这段话，心中又充满了力量！
```

分析：

第 2～5 行：定义了一个 remember 函数，其作用是输出“拼搏到无能为力，坚持到感动自己！”。

第 7 行：main 函数开始。

第 8 行：执行 main 函数第 1 行语句，输出“疲惫的时候，总有个声音在呼喊！”。

第 9 行：调用 remember 函数。main 函数暂停执行，而转去执行 remember 函数，因此程序跳转到第 2 行去执行 remember 函数，执行的结果是输出了“拼搏到无能为力，坚持到感动自己！”。

第 10 行：remember 函数执行完毕后，继续执行 main 函数，输出“想起这段话，心中

又充满了力量!”。

3.1.2 函数的声明

在3.1.1节中,自定义的函数是放在主调函数的前面,有读者可能会提出是否可以把自定义函数放到主调函数的后面,接下来通过一个案例来演示这种情形,具体如例3-2所示。

例 3-2

```
1  #include <stdio.h>
2  int main()
3  {
4      output();
5      return 0;
6  }
7  void output()
8  {
9      printf("做真实的自己!");
10  }
```

分析:

该程序在编译时不通过,将output函数放在main函数的后面定义,结果导致main函数不知道有output这个函数,因此在main函数中调用output函数失败(第4行)。解决办法就是在main函数的前面声明output函数,修改例3-2代码,具体如例3-3所示。

例 3-3

```
1  #include <stdio.h>
2  void output();
3  int main()
4  {
5      output();
6      return 0;
7  }
8  void output()
9  {
10     printf("做真实的自己!");
11 }
```

输出:

```
做真实的自己!
```

分析:

第2行:声明了一个output函数,这样编译器就知道会有一个output函数存在。需要注意的是,函数声明仅仅包括函数定义的头部,不包括后面的大括号与函数体。函数

声明需要用一个英文分号作为结尾。

第 5 行：调用 output 函数，因为前面已经声明了 output 函数，所以成功调用。

第 8～11 行：output 函数的定义。

3.2 有参函数

假如要实现一个执行相加运算的函数，那么就需要给函数添加两个参数，示例代码如下：

```
int add(int x,int y)
{
    return x + y;
}
```

小括号里面不再为空，而是多了两个参数——x 和 y，参数之间用逗号隔开，这里参数的类型为 int，表示 x 和 y 是用来保存整数的。

因此，可以在调用 add 函数时传递两个整数：

```
add(23,11);
```

当调用 add 函数时，23 会传递给 x，由 x 来保存，11 会传递给 y，由 y 来保存。这里需要注意的是，add 函数在调用之前，x 和 y 是不存在的，只有在调用 add 函数，并为 x 和 y 传递了 23 和 11 时，系统才会为 x 和 y 分配内存，而一旦调用结束，系统又会立即释放 x 和 y 所占的内存，所以 x 和 y 不是实际存在的参数，而是形式上存在的参数，又叫形式参数。接下来通过一个案例来演示函数的相加运算，具体如例 3-4 所示。

例 3-4

```
1  #include <stdio.h>
2  int add(int x,int y)
3  {
4      return x + y;
5  }
6  int main()
7  {
8      int sum;
9      sum = add(23,11);
10     printf("%d\n",sum);
11     return 0;
12 }
```

输出：

```
34
```

分析：

第2～5行：定义了一个add函数，该函数有两个类型为int的形式参数x和y，即add函数可以接收两个整数，函数的功能是对接收的两个整数做加法运算，并返回运算结果(第4行)。

第9行：调用add函数，并传递给add函数两个整数——23和11，这样会暂时终止main函数的执行，而转去执行add函数，因此程序跳到第2行，将23和11传递给形式参数x和y，这样x保存了23，y保存了11，第4行执行x+y，得出结果34，关键字return返回这个结果并终止add函数的执行。add函数执行完毕后，会继续执行main函数，因此程序又跳回到第9行，开始执行赋值运算符，将add函数的返回值34赋给左侧的变量sum。

第10行：输出sum的值，即23+11的结果34。

3.3 形式参数与实际参数

形式参数是指在定义函数时函数名后小括号内的变量，简称形参。形参的本质是变量，变量的值是可以改变的，示例代码如下：

```
int i;
```

该行定义了一个变量i，如果把3通过赋值号赋值给变量i，那么i值就是3，因此变量的值不是固定不变的。但对于数字3来说，它是恒定不变的，无法改变数字3的值。

由于变量的这一特性，它常常用来表示不确定值的量，因此变量也可用作函数的参数(函数在调用之前，参数的值都是不确定的)，示例代码如下：

```
int mul(int x, int y)
{
    return x * y;
}
```

这里定义了一个函数mul，它有两个参数x和y，这两个参数都是变量，函数在调用之前，参数的值都是不确定的，因此系统是不会为这两个参数分配内存的，只有调用了函数，并将确切的数值传递给这两个参数的时候，系统才会为这两个参数分配内存，示例代码如下：

```
mul(1, 10);
```

这里调用了mul函数，并为两个形参传递了确切的数值1和10，这样系统才会为形参分配内存，用分配好的内存来保存数值1和10。而当函数调用结束后，系统又会释放形参所占用的内存，因此这样的参数实际上是不存在的，它只是在形式上存在，称为形式参数，简称形参。

实际参数就是在调用 mul 函数时传递的 1 和 10，它们是实际存在的，确切的数值，简称实参。实参可以是常量、变量或表达式，因此可以这样来调用 mul 函数，示例代码如下：

```
int a = 3;
mul(a, 10);
```

第 2 行调用 mul 函数，为其传递了两个实参；第 1 个实参是变量 a，第二个参数是 10，假如不给 i 初始化一个值，那么 Visual C++ 6.0 就会发出警告：

```
warning C4700: local variable 'a' used without having been initialized
```

该警告提示使用了未初始化的局部变量 a，接下来通过一个案例来演示形参和实参的使用，具体如例 3-5 所示。

例 3-5

```
1  #include <stdio.h>
2  int mul(int x, int y)
3  {
4      return x * y;
5  }
6  int main()
7  {
8      int a = 3;
9      printf("%d\n",mul(a, 10));
10     return 0;
11 }
```


```
30
```

第 2 行：定义了一个 mul 函数，它有两个参数 x 和 y，由于这两个参数没有确定的值，并且只在函数调用时才占用内存空间，函数调用前或调用后都不占用内存，因此它们都是形式参数。

第 9 行：为 mul 函数传递两个参数——a 和 10，这两个参数有确切的值(3 和 10)，因此是实际参数。

形参与实参的类型必须一致，否则就会出问题，接下来通过一个案例来演示形参和实参类型不同时的情况，具体如例 3-6 所示。

例 3-6

```
1  #include <stdio.h>
2  int add(int x,int y)
3  {
4      return x + y;
5  }
6  int main()
7  {
8      double d1 = 1.1;
9      double d2 = 2.2;
10     double sum = add(d1,d2);
11     printf("%f\n",sum);
12     return 0;
13 }
```

输出：

```
3.000000
```

该程序在 Visual C++ 6.0 下执行会出现 4 条警告。

```
warning C4244:'function':conversion from 'double' to 'int',possible loss of data
warning C4244:'function':conversion from 'double' to 'int',possible loss of data
warning C4761:integral size mismatch in argument;conversion supplied
warning C4761:integral size mismatch in argument;conversion supplied
```

分析：

C4244 号报警：'函数'：将 double 转换为 int 可能会丢失数据。因为 double 类型的值是带有小数点的值，转换成 int 类型的值时会自动舍去小数点后面的位数，所以可能丢失一部分数据（丢失小数点后面的数据）。

C4761 号报警是因为参数的整型大小不匹配，这 4 条警告都是第 10 行引起的，该行在调用 add 函数时，传递的实参与形参的类型不匹配。

第 2 行：add 函数的定义，该函数有两个形式参数——x 和 y，参数类型为 int。

第 8～9 行：定义两个 double 型变量 d1 和 d2，将 d1 初始化为 1.1，将 d2 初始化为 2.2。

第 10 行：调用 add 函数，将实际参数 d1 和 d2 的值传递给该函数的形式参数 x 和 y，由于形式参数和实际参数的类型不匹配，进行了默认转换，将 double 型数值 1.1 和 2.2 小数点后面的位数抛弃，变成 1 和 2，然后传递给 int 型变量 x 和 y，结果执行的是 1 和 2 的相加操作，所以输出 3.000000。

小结：

实参一定要与形参的类型一致。

3.4 函数的返回值

函数执行后一定要有作用，不是完成输入、输出等工作，就是进行一些计算，计算必然会产生一个结果，而结果可以通过返回值来得到，返回值可以是整数，也可以是字符，示例代码如下：

```
int sub(int x, int y)
{
    return x - y;
}
```

该函数用来计算两个整数的差，因为是求差，所以必须将计算的结果返回。sub 前面的 int 代表返回值的类型，也就是说，返回值必须是个整数，如果不是整数，编译器就会发出警告。返回值是 x－y 的差值，关键字 return 将这个差值返回到主调函数中调用 add 函数的位置处，示例代码如下：

```
int val = sub(10, 1);
```

赋值运算符的右侧调用了 sub 函数，sub 函数执行完毕后，将 10 与 1 相减的结果 9 通过 return 关键字返回到赋值运算符的右侧。这样，sub 函数执行完毕后，赋值运算符的右侧会变成 9，上条语句等价于下条语句：

```
int val = 9;
```

接下来通过一个案例来演示函数返回值的用法，具体如例 3-7 所示。

例 3-7

```
1  #include <stdio.h>
2  int sub(int x, int y)
3  {
4      return x - y;
5  }
6  int main()
7  {
8      int val;
9      val = sub(10, 1);
10     printf("val = %d\n", val);
11     return 0;
12 }
```

输出：

```
val = 9
```

分析：

第 2～5 行：定义了一个 sub 函数，该函数有两个类型为 int 型的参数 x 和 y，即 sub 函数可以接收两个整数，函数的功能是实现两个整数求差并把计算结果返回。

第 9 行：调用 sub 函数，并传递给 sub 函数两个整数 10 和 1，此时程序会暂时终止 main 函数的执行，而跳到第 2 行，将 10 和 1 传递给参数 x 和 y，这样 x 的值就为 10，y 的值就为 1，第 4 行执行 x－y，得出计算结果 9，关键字 return 将计算结果返回。sub 函数执行完毕后，会接着执行 main 函数，因此程序又跳回到第 9 行，开始执行赋值运算符，将 sub 函数的返回值 9 赋给左侧的变量 val，具体如图 3.1 所示。

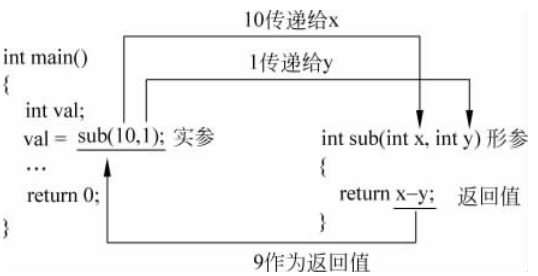


图 3.1　函数调用过程

函数可以返回一个值，当然也可以不返回任何值。如果不想让函数返回任何值，可以将函数的返回值类型定义为 void，示例代码如下：

```
void output()
{
    printf("做真实的自己,用良心做教育!");
}
```

该函数的作用仅仅是输出一条指定文字，因此没有必要给它提供返回值，此时函数返回值类型为 void。

注意：

函数返回值的类型取决于定义函数时指定的函数类型，而不是 return 语句中表达式值的类型。

3.5　printf 函数与 scanf 函数

3.5.1　printf 函数

前面章节中使用到了 printf 函数，它是标准的输出函数，输入与输出功能不是 C 语言的组成部分，而是 C 语言函数库中的标准输入输出函数实现的，因此在使用 printf 函数时要加上 #include < stdio.h >，考虑到读者的现有水平，这里只介绍它的使用方法，接下来通过一个案例来演示 printf 函数的使用，具体如例 3-8 所示。

例 3-8

```
1  #include <stdio.h>
2  int main()
3  {
4      printf("不知不觉喜欢上了C语言\n");
5      printf("%d + %d = %d\n", 1, 10, 1+10);
6      return 0;
7  }
```

输出：

```
不知不觉喜欢上了C语言
1 + 10 = 11
```

分析：

第 4 行：调用 printf 函数，将“不知不觉喜欢上了 C 语言”这几个字符输出到控制台上，字符的结尾跟了一个格式符"\n"，作用是换行，由于它执行的是换行的功能，因此又称为换行符。

第 5 行：调用 printf 函数，输出一则加法运算式，注意%d 是占位符，其中%告诉程序将一个变量或数据在这里输出，d 告诉程序输出的是一个十进制整型变量或一个整数，它对应双引号后面的形式参数，如第 1 个%d 对应着 1，第 2 个%d 对应着 10，第 3 个%d 对应着 1+10，程序运行首先计算出赋值运算符右侧表达式 1+10 的值，再按照以上设定的格式进行输出。

3.5.2 scanf 函数

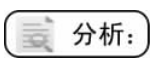
与输出函数 printf 对应的是输入函数 scanf，与 printf 函数一样，scanf 函数也被声明在头文件 stdio.h 里，因此在使用 scanf 函数时要加上#include <stdio.h>。考虑到读者的现有水平，这里也只介绍它的使用方法，接下来通过一个案例来演示 scanf 函数的使用，具体如例 3-9 所示。

例 3-9

```
1  #include <stdio.h>
2  int main()
3  {
4      int i;
5      scanf("i = %d", &i);
6      printf("i = %d", i);
7      return 0;
8  }
```

```
i = 10
```

```
i = 10
```



第 4 行：定义了一个整型变量 i，用它来保存一个整数。

第 5 行：调用 scanf 函数将用户输入的数值保存到变量 i 中，%d 表示这里将用一个整型变量来保存用户输入的整数，它对应着逗号分隔符后面的变量 i，注意 i 前面多了一个符号，这是个“取地址运算符”，&i 用来获取 i 在内存中的地址，获得了 i 的地址后，才能把用户输入的数据直接储存到变量 i 中。此处需要注意输入数据的格式与双引号里面的格式必须一一对应，才能得到正确的输入。

第 6 行：调用 printf 函数将变量 i 的值输出到屏幕上。

读者可能会疑惑：printf 函数不用获取变量的地址就可打印，但 scanf 函数一定要获取变量的地址才可存储。两者的区别就好比小千要借阅小锋的图书，小锋就复印一份给小千（小千不用获得小锋图书的地址），但是假如小千想要修改小锋的图书，那么再用这种方法，修改的将是复印的图书，而不是小锋的图书，因此必须先找到小锋的图书（获得小锋图书的地址），才能对小锋的图书进行修改。

同理，printf 函数仅仅是读取变量的值，不用改变它的值，因此系统只需要将变量复制一份，然后将复制好的变量传递给 printf 函数即可，但是 scanf 函数要修改变量的值，那么再用这种方法，修改的将是复制的变量的值，而不是原始变量的值，因此必须获得原始变量的地址，才能对它进行修改。

3.6 putchar 函数与 getchar 函数

3.6.1 putchar 函数

字符输出函数 putchar 在头文件 stdio.h 中声明，它的作用是向标准输出设备（显示器、打印机、扬声器、指示灯、磁盘或光盘等）中输出一个字符，接下来通过一个案例来演示字符输出函数的使用，具体如例 3-10 所示。

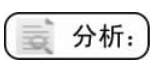
例 3-10

```
1  #include <stdio.h>
2  int main()
3  {
4      char a, b, c;
```

```
5       a = 'Q';
6       b = '\n';
7       c = 'F';
8       putchar(a);
9       putchar(b);
10      putchar(b);
11      putchar(c);
12      return 0;
13  }
```


```
Q


F
```



第 4 行：定义 3 个用来保存字符的变量 a、b 和 c。

第 5 行：用变量 a 保存字符 Q。

第 6 行：用变量 b 保存换行符。

第 7 行：用变量 c 保存字符 F。

第 8 行：调用 putchar 函数将变量 a 的值输出到控制台，结果输出了字符 Q。

第 9～10 行：调用 putchar 函数将变量 b 的值输出到控制台，结果进行了换行。

第 11 行：调用 putchar 函数将变量 c 的值输出到控制台，结果输出了字符 F。

3.6.2 getchar 函数

字符输入函数 getchar 在头文件 stdio.h 中声明，它的作用是从标准输入设备中获取一个字符，getchar 函数没有参数，使用时可以直接调用。此外，当用户输入字符并按 Enter 键时，输入的字符才送到输入缓冲区（内存中用来暂存输入内容的一块特殊区域），因此 getchar 函数是用来获取输入缓冲区中的一个字符。接下来通过一个案例来演示字符输入函数的使用，具体如例 3-11 所示。

例 3-11

```
1   #include <stdio.h>
2   int main()
3   {
4       char c;
5       c = getchar();
6       putchar(c);
7       c = getchar();
8       putchar(c);
9       return 0;
10  }
```

输入：

```
A(回车)
```

输出：

```
A
```

分析：

第 5 行：调用 getchar 函数，获得用户从键盘上输入的字符 A，并将该字符返回，变量 c 保存了返回的字符 A。

第 6 行：调用 putchar 函数将变量 c 保存的字符 A 输出到控制台。

第 7 行：在输入时，字符 A 和 Enter 都存储在输入缓冲区，再次调用 getchar 函数，则从输入缓冲区中提取的是换行符'\n '。

第 8 行：调用 putchar 函数输出变量 c 的值，结果光标移动到下一行的开头。

3.7 变量的作用域

通过前面的学习，发现变量既可以定义在函数内，也可以定义在函数外。定义在不同位置的变量，其作用域也是不同的。C 语言中的变量，按作用域范围可分为局部变量和全局变量。

3.7.1 局部变量

局部变量就是在函数内部声明的变量，它只在函数内有效，也就是说，只能在本函数内使用它。此外，局部变量只有当它所在的函数被调用时才会被使用，而当函数调用结束时局部变量就会失去作用。接下来通过一个案例来演示局部变量的使用，具体如例 3-12 所示。

例 3-12

```
1  #include <stdio.h>
2  void say()
3  {
4      char c = 'a';
5  }
6  int main()
7  {
8      printf("c 的值为：%d\n", c);
9      return 0;
10 }
```

输出：

```
编译报错：
错误:error C2065:'c':undeclared identifier
```

分析：

第 4 行：在 say 函数内部定义一个变量 c，这个变量是局部变量，它只在 say 函数中有效。

第 8 行：main 函数试图输出 c 的值，由于 c 只在 say 函数中有效，在 main 函数中无效，因此编译器报错。

另外，关于局部变量还有以下 4 点说明：

- 在 main 函数中定义的变量也是局部变量，只能在 main 函数中使用，main 函数也是一个函数，与其他函数地位平等。
- 形参变量、在函数体内定义的变量都是局部变量。实参给形参传值的过程也就是给局部变量赋值的过程。
- 可以在不同的函数中使用相同的变量名，它们表示不同的数据，分配不同的内存，互不干扰，也不会发生混淆。
- 在语句块中也可定义变量，它的作用域只限于当前语句块。

3.7.2 全局变量

与局部变量相对应的是全局变量。它是指在所有函数外部定义的变量。它的有效范围为从定义开始到程序结束。接下来通过一个案例来演示全局变量的使用，具体如例 3-13 所示。

例 3-13

```
1  #include <stdio.h>
2  int x = 1;
3  void output()
4  {
5      x = 3;
6      printf("output 函数中 x = %d\n", x);
7  }
8  int main()
9  {
10     printf("main 函数中 x = %d\n", x);
11     x = 2;
12     printf("main 函数中 x = %d\n", x);
13     output();
14     printf("main 函数中 x = %d\n", x);
15     return 0;
16 }
```

输出：

```
main 函数中 x = 1
main 函数中 x = 2
output 函数中 x = 3
main 函数中 x = 3
```

分析：

第 2 行：定义了一个全局变量 x。

第 10 行：在 main 函数中打印全局变量的值为 1。

第 11 行：在 main 函数中修改全局变量的值为 2。

第 13 行：在 main 函数中调用 output 函数，在 output 函数中修改全局变量的值为 3。

第 14 行：output 函数调用结束后，打印全局变量的值为 3。

小结：

全局变量可以被所有的函数所共享，因此它可以作为各个函数之间相互沟通的渠道。它的缺点也正在于此。因为共享，数据容易被修改，变量名容易重复，要始终占据内存。

因此，在使用全局变量时应注意以下 3 点：

- 尽可能使名字易于理解，而且不能太短，避免重名发生。
- 避免使用一些占用内存空间比较大的全局变量，以节省开销。
- 为避免全局变量被误修改，应尽量在所有修改全局变量的函数前添加注释，用来说明该函数都修改了哪些全局变量，修改的目的是什么，修改值又是多少。

3.7.3 局部变量与全局变量

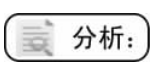
接下来通过一个案例来演示局部变量屏蔽全局变量的情形，具体如例 3-14 所示。

例 3-14

```
1  #include <stdio.h>
2  int x = 1;
3  void output()
4  {
5      int x = 3;
6      printf("output 函数中局部变量 x = %d\n", x);
7  }
8  int main()
9  {
10     printf("全局变量 x = %d\n", x);
11     {
12         int x = 2;
```

```
13          printf("块中局部变量 x = %d\n", x);
14      }
15      output();
16      x = 4;
17      printf("main 函数中修改全局变量的值 x = %d\n", x);
18      return 0;
19  }
```


```
全局变量 x = 1
块中局部变量 x = 2
output 函数中局部变量 x = 3
main 函数中修改全局变量的值 x = 4
```



第 2 行：在函数外定义了一个全局变量 x，并将它的值初始化为 1。它的作用域从第 2 行开始到第 19 行结束。

第 5 行：在 output 函数中定义了一个局部变量 x，并将它的值初始化为 3。它的作用域从第 5 行开始到第 7 行结束。

第 10 行：在 mian 函数中输出全局变量 x 的值 1。

第 12 行：在块中定义一个局部变量 x，并将它的值初始化为 2。它的作用域从第 12 行开始到第 14 行结束。

第 13 行：在块中输出局部变量 x 的值 2，说明在块中局部变量 x 屏蔽了全局变量 x。

第 15 行：在 main 函数中调用 output 函数，输出 x 的值为 3，这是 output 函数中局部变量 x 的值，说明在 output 函数中局部变量 x 屏蔽了全局变量 x。

第 16 行：在 main 函数中修改全局变量 x 的值为 4。

3.8 本章小结

通过本章的学习，能够掌握 C 语言初级函数的使用，重点要了解的是在实际开发中，要完成一个复杂的程序，可将一个复杂的程序简化成若干个函数来实现。

3.9 习 题

1. 填空题

(1) 用来结束函数并返回函数值的是_____关键字。

(2) 按用户指定的格式从标准输入设备上把数据输入到指定变量中的函数是_____。

(3) 函数调用语句 function((exp1,exp2),18);中含有的实参个数为_____。

(4) 从标准输入设备中获取一个字符的函数是_____。

(5) 按作用域范围不同可将变量分为局部变量和_____变量。

2. 选择题

(1) C 语言中函数返回值的类型是由(　　)决定。

A. return 语句中的表达式类型

B. 调用函数的主调函数类型

C. 调用函数时临时

D. 定义函数时所指定函数返回值类型

(2) 在该函数体内说明语句后的复合语句中定义了一个变量,则该变量(　　)。

A. 为全局变量,在本文件内有效

B. 为局部变量,只在该函数内有效

C. 定义无效,为非法变量

D. 为局部变量,只在该复合语句内有效

(3) 定义一个 void 型函数意味着调用该函数时,函数(　　)。

A. 通过 return 返回一个指定值　　B. 返回一个系统默认值

C. 没有返回值　　D. 返回一个不确定的值

(4) 以下对函数声明错误的是(　　)。

A. float add(float a,b);　　B. float add(float b,float a);

C. float add(float,float);　　D. float add(float a,float b);

(5) 以下关于函数的叙述中不正确的是(　　)。

A. C 程序是函数的集合

B. 被调函数必须在 main 函数中定义

C. 函数的定义不能嵌套

D. 函数的调用可以嵌套

3. 思考题

(1) 请简述全局变量和局部变量有什么区别。

(2) 请简述全局变量有哪些缺点。

(3) 请简述函数形式参数与实际参数有什么区别。

(4) 请简述 putchar、getchar 它们各自的作用是什么。

4. 编程题

(1) 编写一个函数实现输出“Welcome to QianFeng!”,要求此函数放在主函数之后。

(2) 编写一个求和函数,主函数从键盘接受两个整数并输出求和结果。