

C语言与程序设计 The C Programming Language & Program Design

第5章 函数与程序结构

华中科技大学计算机学院
甘早斌

3/27/2015 华中科技大学计算机学院 甘早斌 1/81

本章的教学目的

- 掌握结构化编程方法和C程序的一般结构
- 熟悉函数的机制，包括：
 - 函数定义
 - 函数声明
 - 函数调用
 - 变量的存储类型
 - 参数数目可变的函数

3/27/2015 华中科技大学计算机学院 甘早斌 2/81

本章目录

- 5.1 C程序的一般结构
- 5.2 函数的定义与函数原型
- 5.3 函数调用与参数传递
- 5.4 作用域与可见性
- 5.5 存储类型

3/27/2015 华中科技大学计算机学院 甘早斌 3/81

5.1 C程序的一般结构

- 5.1.1 结构化程序设计
- 5.1.2 蒙特卡罗模拟：猜数游戏
- 5.1.3 C程序的结构

3/27/2015 华中科技大学计算机学院 甘早斌 4/81

5.1.1 结构化程序设计 1/8

- 结构化程序设计(structured programming)的诞生和发展
 - 结构化程序设计方法的起源来自对GOTO语句的认识和争论
 - 结构化程序设计在1960年代开始发展，科拉多·伯姆及朱塞佩·贾可皮尼伯姆于1966年5月在《Communications of the ACM》期刊发表论文，说明任何一个有goto指令的程序，可以改为完全不使用goto指令的程序
 - 1968年艾兹赫尔·戴克斯特拉提出著名的论文《Go To Statement Considered Harmful GOTO》
 - 正方观点：
 - 在块和进程的非正常出口处往往需要用GOTO语句，使用GOTO语句会使程序执行效率较高
 - 在合成程序目标时，GOTO语句往往是有用的，如返回语句用GOTO

3/27/2015 华中科技大学计算机学院 甘早斌 5

5.1.1 结构化程序设计 2/8

- 反方观点：
 - GOTO语句是有害的，是造成程序混乱的祸根，程序的质量与GOTO语句的数量呈反比，应该在所有高级程序设计语言中取消GOTO语句
 - 取消GOTO语句后，程序易于理解、易于排错、容易维护，容易进行正确性证明
- 作为争论的结论，1974年Knuth发表了令人信服的总结并证实：
 - (1) GOTO语句确实有害，应当尽量避免
 - (2) 完全避免使用GOTO语句也并非是个明智的方法，有些地方使用GOTO语句，会使程序流程更清楚、效率更高
 - (3) 争论的焦点不应该放在是否取消GOTO语句上，而应该放在用什么样的程序结构上。其中最关键的是，应在以提高程序清晰性为目标的结构化方法中限制使用GOTO语句

3/27/2015 华中科技大学计算机学院 甘早斌 6

唐纳德·克努特Donald Ervin Knuth


□ 1938年1月10日生于美国威斯康星州(Wisconsin)。

□ 斯坦福大学计算机科学系荣誉退休教授，排版软件TeX和字型设计系统Metafont发明人。

□ 所著描述基本算法与数据结构的巨作《计算机程序设计的艺术》被《美国科学家》杂志列为20世纪最重要的12本物理科学类专著之一，与爱因斯坦《相对论》、狄拉克《量子力学》、理查·费曼《量子电动力学》等经典比肩而立。

□ 荣获1974年度的图灵奖（时年36岁）。

□ 他是最年轻图灵奖获得者纪录的保持者。



3/27/2015

华中科技大学计算机学院 甘早斌

5.1.1 结构化程序设计3/8

□ 结构化程序设计的基本思想

■ 结构化程序设计是一种解决问题的策略，单入口单出口的控制结构；

■ 采用自顶向下、逐步求精及模块化的程序设计方法；

■ 使用三种基本控制结构构造程序，任何程序都可由顺序、选择、循环三种基本控制结构构造；

■ 结构化程序设计主要强调的是程序的易读性。

3/27/2015

华中科技大学计算机学院 甘早斌

8

5.1.1 结构化程序设计4/8

□ 基本结构

■ 顺序结构

□ 顺序结构表示程序中的各操作是按照它们出现的先后顺序执行的。

■ 选择结构

□ 选择结构表示程序的处理步骤出现了分支，它需要根据某一特定的条件选择其中的一个分支执行。

□ 选择结构有单选择、双选择和多重选择三种形式。

■ 循环结构

□ 循环结构表示程序反复执行某个或某些操作，直到某条件为假（或为真）时才可终止循环。在循环结构中最主要的是：什么情况下执行循环？哪些操作需要循环执行？循环结构的基本形式有两种：当型循环和直到型循环。

3/27/2015

华中科技大学计算机学院 甘早斌

9

5.1.1 结构化程序设计5/8

■ 当型循环

□ 表示先判断条件，当满足给定的条件时执行循环体，并且在循环终端处流程自动返回到循环入口；如果条件不满足，则退出循环体直接到达流程出口处。

□ 因为是“当条件满足时执行循环”，即先判断后执行，所以称为当型循环。

■ 直到型循环

□ 表示从结构入口处直接执行循环体，在循环终端处判断条件，如果条件不满足，返回入口处继续执行循环体，直到条件为真时再退出循环到达流程出口处，是先执行后判断。

□ 因为是“直到条件为真时为止”，所以称为直到型循环。

3/27/2015

华中科技大学计算机学院 甘早斌

10

5.1.1 结构化程序设计6/8

□ 结构化程序设计是进行以模块功能和处理过程设计为主的详细设计的基本原则：

■ 自顶向下

□ 程序设计时，应先考虑总体，后考虑细节；先考虑全局目标，后考虑局部目标。不要一开始就过多追求众多的细节，先从最上层总目标开始设计，逐步使问题具体化。

■ 逐步细化

□ 对复杂问题，应设计一些子目标作为过渡，逐步细化。

■ 模块化设计

□ 一个复杂问题，肯定是由若干稍简单的问题构成。

□ 模块化是把程序要解决的总目标分解为子目标，再进一步分解为具体的小目标，把每一个小目标称为一个模块。

3/27/2015

华中科技大学计算机学院 甘早斌

11

5.1.1 结构化程序设计7/8

□ 结构化程序设计的特点

■ 结构化程序中的任意基本结构都具有唯一入口和唯一出口，并且程序不会出现死循环

■ 在程序的静态形式与动态执行流程之间具有良好的对应关系

■ 任何算法功能都可以通过由程序模块组成的三种基本程序结构的组合：顺序结构、选择结构和循环结构来实现

3/27/2015

华中科技大学计算机学院 甘早斌

12

2

5.1.1 结构化程序设计

8/8

□ 结构化程序设计的优点

- ①整体思路清楚，目标明确，提高软件的可重用性
- ②编写的程序出结构良好、易于管理、修改和调试
- ③设计工作中阶段性非常强，有利于系统开发的总体管理和控制，方便于多人分工合作完成程序的编制
- ④增强了程序的可读性、可维护性和可扩充性，在系统分析时可以诊断出原系统中存在的问题和结构上的缺陷

□ 结构化程序设计的缺点

- ①用户要求难以在系统分析阶段准确定义，致使系统在交付使用时产生许多问题
- ②用系统开发每个阶段的成果来进行控制，不能适应事物变化的要求
- ③系统的开发周期长

3/27/2015

华中科技大学计算机学院 甘卓斌

13

5.1.2 蒙特卡罗模拟：猜数游戏

□ 模拟算法是最基本的算法，如，编程实现抛硬币、掷骰子和玩牌等现实世界中的随机事件要用模拟算法

□ 在程序设计中，可使用随机数函数来模拟现实中不可预测情况，这称为蒙特卡罗模拟

□ 随机数以其不确定性和偶然性等特点在很多地方都有具体的用处。比如：

- 软件测试中，用于产生具有普遍意义的测试数据
- 在加密系统中产生密钥
- 在网络中生成验证码等

3/27/2015

华中科技大学计算机学院 甘卓斌

14

□ 在C语言中，用rand函数生成随机数，该函数称为随机数发生器，该发生器从称为种子（一个无符号整型数）的初始值开始用确定的算法产生随机数

□ 伪随机数

- 通过种子产生第一个随机数后，后续的随机序列也就是确定的了，这种依靠计算机内部算法产生的“随机”数称为伪随机数

□ 真随机数

- 随机数的产生依赖于种子，为了使程序在反复运行时能产生不同的随机数，必须改变这个种子的值，这称为初始化随机数发生器，由函数srand来实现

3/27/2015

华中科技大学计算机学院 甘卓斌

15

□ 【例5.1】编写一个猜数的游戏程序

- 在这个游戏中，计算机产生一个1到1000之间的随机数，并把该数作为要猜的数。
- 玩游戏者输入所猜的数，如果猜得不正确，继续猜直到正确为止，同时计算游戏者猜数的次数。
- 为了帮助游戏者一步一步得到正确答案，程序会不断地发出信息“Too high”或“Too low”。最后，程序向游戏者显示游戏结果。

3/27/2015

华中科技大学计算机学院 甘卓斌

16

□ 自顶向下的分解问题：

- 既然是一个游戏程序，就应该允许玩家反复玩多次，直到不想玩为止。同时，将玩一次游戏的任务分解成以下两个子任务：
 - (1) 计算机产生一个1到1000的随机数供游戏者猜测；
 - (2) 游戏者猜数，直至猜对。

3/27/2015

华中科技大学计算机学院 甘卓斌

17

□ 主程序结构

```
do {
    计算机产生一个1到1000的随机数
    游戏者猜数，直至猜对
    继续玩吗
} while (继续);
```

3/27/2015

华中科技大学计算机学院 甘卓斌

18

□ 自顶向下的分解子任务 (1)

- ① 调用标准库函数rand产生一个随机数;
- ② 将这个随机数限制在1~1000之间。
- 利用函数, 可以实现程序的模块化, 把程序中常用的一些算法或操作编成通用的函数, 以供随时调用, 大大简化主函数的流程, 使程序设计简单和直观, 提高程序的易读性和可维护性。
- 把任务1设计成一个独立的函数, 用函数GetNum(void) 来实现。

3/27/2015

华中科技大学计算机学院 甘卓斌

19

□ 函数 int GetNum(void)

```

1.  /*=====*/
2.  函数名称: GetNum
3.  函数功能: 产生一个1到MAX_NUMBER之间的随机数, 供游戏者猜测。
4.  函数参数: 无
5.  函数返回值: 返回产生的随机数
6.  /*=====*/
7.  int GetNum(void)          /* 注意: 后面无分号 */
8.  {                          /* 函数开始的标志 */
9.      int x;
10.     printf("A magic number between 1 and %d has been chosen.\n", MAX_NUMBER);
11.     x=rand();              /* 调用标准库函数rand产生一个随机数 */
12.     x= x % MAX_NUMBER + 1; /*将这个随机数限制在1~MAX_NUMBER之间 */
13.     return(x);             /* 返回这个随机数给调用者, */
14. }                          /* 函数结束的标志 */

```

3/27/2015

华中科技大学计算机学院 甘卓斌

20

□ 加注释——程序编程规范

- 在函数的顶端用“/*.....*/”格式包含的部分是函数头部注释, 包括函数名称、函数功能、函数参数、函数返回值等内容
- 如有必要还可增加作者、创建日期、修改记录(备注)等相关项目。
- 虽然函数头部注释在语法上不是必需的, 但可以提高程序的质量和可维护性, 在程序设计时要遵从这一编程规范。

□ int GetNum(void)

- GetNum是函数名, 其后的void说明函数调用时不接收任何参数, 即没有入口参数。
- 函数执行完应该返回所产生的随机数, 即该随机数是函数的出口参数, 函数名前的int说明出口参数的类型为整型。

3/27/2015

华中科技大学计算机学院 甘卓斌

21

□ int __cdecl __MINGW_NOTHROW rand (void);

- rand()是stdlib.h中的一个函数
- 返回一个非负并且不大于常量RAND_MAX的随机整数, RAND_MAX的值取决于计算机系统。
- #define RAND_MAX 0x7FFF

- MAX_NUMBER是用#define定义的符号常量, 其值为1000。当执行return语句时, 其后表达式的值被带回到调用函数中。

3/27/2015

华中科技大学计算机学院 甘卓斌

22

□ 自顶向下的分解子任务 (2)

/* 子任务2的程序段 */

```

for(;;) {
    输入猜测的数
    if (猜对了) 结束
    else if (小了) 输出太小的提示
    else 输出太大的提示
}

```

- 将任务 (2) 也设计成一个独立的函数GuessNum

3/27/2015

华中科技大学计算机学院 甘卓斌

23

□ 函数 void GuessNum(int x)

- GuessNum是函数名, 括号里的“int x”说明该函数有一个入口参数x, 其类型为整型, 表示被猜测的神秘数, 游戏者如果猜对就直接屏幕输出猜的次数, 无具体值返回, 即函数无出口参数, 所以将函数值的类型说明为void。
- 函数体内有1条for(;;)循环语句, 一直循环到执行return语句时结束。
- [源程序\ex5_1.c](#)

3/27/2015

华中科技大学计算机学院 甘卓斌

24

```

1.  /*=====*/
2.  函数名称: GuessNum
3.  函数功能: 游戏者根据提示反复猜数, 直至猜对为止, 输出猜数的次数。
4.  函数参数: x表示被猜的数
5.  函数返回值: 无
6.  /*=====*/
7.  void GuessNum(int x)
8.  {
9.      int guess, counter = 0;
10.     for (;;)
11.     {
12.         printf("guess it: ");
13.         scanf("%d", &guess);
14.         counter++; /* 统计猜的次数 */
15.         if (guess == x) { /* 猜对 */
16.             printf("You guessed the number by %d times!\n", counter);
17.             return;
18.         }
19.         else if (guess < x) { /* 猜小了 */
20.             printf("Too low. Try again.\n");
21.         }
22.         else { /* 猜大了 */
23.             printf("Too high. Try again.\n");
24.         }
25.     }
26. }

```

主函数main

- 用蒙特卡罗法模拟该猜数游戏, 在开始玩游戏前, 需要调用函数srand初始化随机数种子, 可以采用系统时间 (由time函数得到) 作为种子。
- 函数time返回自1970年1月1日以来经历的秒数, 将该秒数赋值给种子变量, 随机数种子会随着运行程序的时间而改变, 因而产生的随机数是不可预见的。
- 函数srand和rand的原型在stdlib.h中, 函数time的原型在time.h中, 需要在源程序的头部包含这两个头文件。
- [源程序ex5_1.c](#)

3/27/2015

华中科技大学计算机学院 甘早斌

26

```

/*=====*/
简要描述: 模拟猜数的游戏程序
规则: 程序产生一个1到1000之间的随机数让游戏者猜,
如果猜得不正确, 程序会不断地发出信息 "Too high"或 "Too low", 继续猜直到正确为止,
输出猜数的次数。
/*=====*/
1.  #include <stdio.h>
2.  #include <stdlib.h>
3.  #include <time.h>
4.  #define MAX_NUMBER 1000
5.  int GetNum (void); /* 函数原型 */
6.  void GuessNum(int); /* 函数原型 */
7.  int main(void)
8.  {
9.      char command;
10.     int magic;
11.     printf("This is a guessing game!\n");
12.     srand(time(NULL)); /* 用系统时间初始化随机数生成器 */
13.     do
14.     {
15.         magic = GetNum(); /* 调用GetNum产生随机数供猜测 */
16.         GuessNum(magic); /* 调用GuessNum猜出这个数 */
17.         printf("Play again? (Y/N) ");
18.         scanf("%1s", &command); /* 询问是否继续 */
19.     } while (command == 'y' || command == 'Y');
20.     return 0;
21. }

```

5.1.3 C程序的结构

- C程序由一个或多个函数组成, 其中有且只有一个main函数, 程序的执行总是从main开始。
- 除main以外的其它函数分两类:
 - 一类是由系统提供的标准函数。
 - 另一类是需要由程序员自己编写的函数(“自定义函数”)。
- 组成一个C程序的各个函数可以编辑成多个C源文件, 每一个C源文件含有一个或多个函数定义。
- 各C源文件中要用到的一些外部变量说明、枚举类型声明、结构类型声明、函数原型和编译预处理指令等可编辑成一个.h头文件, 然后在每个C文件中包含该头文件。

3/27/2015

华中科技大学计算机学院 甘早斌

28

- 每个源文件可单独编译生成目标文件, 组成一个C程序的所有源文件都被编译之后, 由连接程序将各目标文件中的目标函数和系统标准函数库的函数装配成一个可执行C程序。下图显示了C语言程序的基本结构。



3/27/2015

华中科技大学计算机学院 甘早斌

29

5.2 函数的定义与函数原型

- 5.2.1 函数的定义
- 5.2.2 函数的返回值
- 5.2.3 函数的声明

3/27/2015

华中科技大学计算机学院 甘早斌

30

5.2.1 函数的定义

1/3

- 程序中若要使用自定义函数实现所需的功能，需要做三件事：

- ① 按语法规则编写完成指定任务的函数，即定义函数；
- ② 有些情况下在调用函数之前要进行函数声明；
- ③ 在需要使用函数时调用函数。

- 函数定义的一般形式为：

类型名 函数名 (参数列表)

```
{
    声明部分
    语句部分
}
```

3/27/2015

华中科技大学计算机学院 甘卓斌

31

5.2.1 函数的定义

2/3

- 类型名

- 类型名说明函数返回值（即出口参数）的数据类型（简称为函数的类型或函数值的类型），可以是除数组以外的任何类型。
- 当返回值类型为void，函数将不返回任何值。

- 参数列表

- 参数列表说明函数入口参数的名称、类型和个数，它是一个用逗号分隔的变量名及其类型列表，它描述了在调用函数时函数所接收的参数。
- 一个函数可能没有参数，在没有参数的情况下，参数列表说明为void，否则必须明确地列出每一个参数的类型。
- 在参数列表中定义的参数称为形式参数（简称形参），用以强调它作为占位符的角色。

3/27/2015

华中科技大学计算机学院 甘卓斌

32

5.2.1 函数的定义

3/3

- 形参是函数要处理的数据，在函数调用时要将实际值传递给形参。在定义变量时，相同类型的变量可以用一个数据类型关键字定义。但是，和变量定义不同的是，每一个形参都必须有数据类型和名字。

- 例如：

```
□ double power(int x, int n) { ... } /* 正确的函数参数定义 */
□ double power(int x, n) { ... } /* 错误，n必须指定类型 */
□ int GetNum(void) { ... } /* 参数表为空的函数 */
□ GetNum(void) { ... } /* C89中默认返回int，但C99不允许 */
```

- 良好的编程风格是：在每个函数的顶端用“/*.....*/”格式增加函数头部注释。

- 函数命名应选择有意义的名称，以增加程序的可读性，还可避免过多地使用注释。Windows风格函数名用小写字母命名，每个词的第一个字母大写，通常用“动词”或“动词+名词”（动词词组）形式。

3/27/2015

华中科技大学计算机学院 甘卓斌

33

5.2.2 函数的返回值

1/2

- return语句可以是如下两种形式之一：

- (1) return ; /* void函数 */
- (2) return 表达式 ; /* 非void函数 */

- void函数也可以不包含return语句。如果没有return语句，当执行到函数结束的右花括号时，控制返回到调用处，把这种情况称为离开结束。

- 表达式值的类型应该与函数定义的返回值类型一致，如果不相同，就把表达式值的类型自动转换为函数的类型。

3/27/2015

华中科技大学计算机学院 甘卓斌

34

5.2.2 函数的返回值

2/2

- 函数返回的值，程序可以使用它，也可以不使用它

```
while(...) {
    getchar(); /* 返回值不被使用 */
    c=getchar(); /* 返回值被使用 */
    ...
}
```

- 例5.2 写一个函数IsPrime判断整数n是否为素数。如果n是素数，则返回1；如果n不是素数，则返回0。

- 源程序ex5_2.c

- 在一个函数中可以有多个return语句，此种情况下的return语句通常被作为选择语句的子句出现，最终被执行的只是其中的一个(Why?)
- 一旦某个return语句被执行，控制立即返回到调用处，其后的代码不可能被执行。

3/27/2015

华中科技大学计算机学院 甘卓斌

35

```
1. /******
2. 简要描述：输入一个整数，判断整数它是否是素数
3. *****/
4. #include <stdio.h>
5. /******
6. 函数名称：IsPrime
7. 函数功能：判断整数是否是素数
8. 函数参数：第1个参数n为int，表示待判断的整数
9. 函数返回值：如果是素数，则返回1；如果不是素数，则返回0。
10. *****/
11. int IsPrime(int n)
12. {
13.     int k, limit;
14.     if (n==2) return 1;
15.     if ((n % 2) == 0) return 0;
16.     limit=n/2; /* 为何选用n/2? */
17.     for (k=3; k<=limit; k+=2)
18.         if ((n % k) == 0) return 0;
19.     return 1;
20. }
21. int main()
22. {
23.     int x;
24.     printf("input a integer:");
25.     scanf("%d", &x);
26.     if (IsPrime(x)) printf("%d is a prime.\n", x);
27.     else printf("%d is not a prime.\n", x);
28.     return 0;
29. }
```

5.2.3 函数的声明

1/5

□ 1. 函数定义起函数声明的作用

- 要利用函数定义起函数声明的作用，在调用函数之前，必须给出调用函数的函数定义。例如，将main函数的定义放在最后。

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#define MAX_NUMBER 1000
int GetNum(void) { ... }
void GuessNum(int x) { ... }
int main(void) { ... }
```

3/27/2015

华中科技大学计算机学院 甘卓斌

37

5.2.3 函数的声明

2/5

□ 2. 函数原型

- 函数定义出现在函数调用后
- 被调用函数在其它文件中定义
- 必须在函数调用之前给出函数原型

□ 3. 函数原型的一般形式

类型名 函数名 (参数类型表);

- 参数类型表通常是用逗号隔开的形参类型列表，而形参名可以省略。例如，

```
void GuessNum(int);
```

□ 等价于

```
void GuessNum(int x);
```

- 对没有参数的函数，函数原型的参数表必须指定为void

3/27/2015

华中科技大学计算机学院 甘卓斌

38

5.2.3 函数的声明

3/5

□ 函数原型的作用

- 函数原型告诉编译器函数返回值的数据类型，传递给函数的参数个数、参数类型和参数顺序。
- 编译器用函数原型校验函数调用，强制转换传递的参数类型，从而避免错误的函数调用导致的致命运行错误或微妙而难以检测的非致命逻辑错误。例如，

```
printf("%.1f\n", sqrt(4));
```

- (1)包含了math.h，输出2.0
- (2)没有包含math.h，函数调用sqrt(4)不会产生正确的值。

3/27/2015

华中科技大学计算机学院 甘卓斌

39

5.2.3 函数的声明

4/5

□ 4. 遗漏函数原型

- 编译器首次遇到没有声明的函数调用时，将构造一个缺省声明并继续编译：

```
int f(); /* 函数是int类型，但不给出参数表信息 */
```

- (1) 函数类型不是int，给出一个错误提示：
Type mismatch in redeclaration of 'f'
- (2) 函数类型是int，就不给出错误提示，**编译器对参数类型不作检查**，把正确类型的参数传递给函数是程序员的责任。
- 假设函数f只有一个类型为int的参数，函数调用f(2)会产生正确的值，而函数调用f(2.5)将产生错误的运行结果。

3/27/2015

华中科技大学计算机学院 甘卓斌

40

5.2.3 函数的声明

5/5

□ 良好的编程风格

- 在函数调用之前必须给出它的函数定义、函数原型或两者都给出。
- 引入标准头文件的主要原因是它含有函数原型。

3/27/2015

华中科技大学计算机学院 甘卓斌

41

5.3 函数调用与参数传递

□ 5.3.1 函数调用

□ 5.3.2 参数的值传递

3/27/2015

华中科技大学计算机学院 甘卓斌

42

5.3.1 函数调用

1/4

1. 函数调用的形式

函数名 (实参列表)

■ 实参是一个表达式,对于无函数,调用形式:
函数名 ()

■ (1) 作为表达式语句出现。

GuessNum();

putchar(c);

■ (2) 作为表达式中的一个操作数出现。例如:

c=getchar()

magic = GetNum();

■ (3) 作为函数调用的一个实参出现,即嵌套调用。

printf("%10.0f",sqrt(X));

while(putchar (getchar()))!= '#' ; 执行结果?

3/27/2015

华中科技大学计算机学院 甘卓斌

43

5.3.1 函数调用

2/4

2. 函数调用的执行过程

■ 例5.3 编写程序实现如下功能: 分列整齐地显示整数1到10的2至5次幂。
输出结果如下所示:

Int	Square	Cube	Quartic	Quintic
1	1	1	1	1
2	4	8	16	32
3	9	27	81	243
4	16	64	256	1024
5	25	125	625	3125
6	36	216	1296	7776
7	49	343	2401	16807
8	64	512	4096	32768
9	81	729	6561	59049
10	100	1000	10000	100000

■ 源程序'ex5_3.c'

3/27/2015

华中科技大学计算机学院 甘卓斌

44

```

1  /******
2  简要描述: 分列整齐地显示整数1到10的2至5次幂。
3  *****/
4  #include<stdio.h>
5  double power(int,int); /* 函数原型 */
6  int main(void)
7  {
8      int i,j;
9      printf("n%5s%10s%10s%10s%10s\n","Int","Square","Cube","Quartic","Quintic"); /* 显示表头 */
10     for(i=1;j=2;j<=5;j++) /* 分列整齐地显示整数1到10的2至5次幂 */
11     {
12         printf("%5d",i);
13         for(j=2;j<=5;j++) printf("%10.0f",power(i,j));
14         printf("\n");
15     }
16     return 0;
17 }
18
19 /******
20 函数名称: power          函数参数: 第1个参数x为int, 第2个参数n也为int
21 函数功能: 计算x的n次方  函数返回值: 返回x的n次方, 为避免溢出, 将函数值的类型说明为double。
22 *****/
23 double power(int x, int n)
24 {
25     int i;
26     double p;
27     for (i=1;p<=n;i++) p*=x;
28     return p;
29 }

```

5.3.1 函数调用

3/4

程序分析

■ 该程序由主函数main和用户自定义函数power组成。main函数在printf语句中嵌套调用了power函数。

■ 程序从main函数开始执行,当执行到printf("%10.0f",power(i,j));时,首先调用函数power,这时系统为形参x和n分配2或4字节的存储单元,将实参i的值传递给形参x,将实参j的值传递给形参n,然后把程序控制转移到函数power,执行其函数体内的语句,其中的for语句计算xn,并把计算结果保存在变量p中,语句return p;将程序控制返回到调用处(即main函数中的printf("%10.0f",power(i,j));语句),同时把p的值送回,至此,power函数调用执行完毕,系统释放形参所占据的存储单元。程序从main函数调用点继续执行,输出返回的p值。

■ main函数用二重for循环结构重复调用了power函数40次,每次传递给形参的值不同即可计算出题目要求的所有xn值(x=1~10,n=2~5)。

3/27/2015

华中科技大学计算机学院 甘卓斌

46

5.3.1 函数调用

4/4

3. 实参的求值顺序

■ 函数调用中实参之间的逗号是分隔符,它与顺序求值运算符','不同

■ 实参的求值顺序由具体实现确定,有的按从左至右的顺序计算,有的按从右至左的顺序计算

a=1;

power(a,a++)

----从左至右: power(1,1)

----从右至左: power(2,1) (多数)

■ 为了保证程序清晰、可移植,应避免使用会引起副作用的实参表达式。

3/27/2015

华中科技大学计算机学院 甘卓斌

47

5.3.2 参数的值传递

1/2

■ 参数的传递方式是“值传递”,实参的值单向传递给相应的形参。如果实参、形参都是x,被调用函数不能改变实参x的值。

■ 例5.4 改写例5.3来说明值传递概念。

■ 源程序'ex5_4.c'

■ 由于参数是值传递,power函数中的形参n可以当普通的局部变量使用,被用作临时变量,控制for的循环次数,这样就不再需要引入局部变量i,从而使程序更简洁。

■ 主函数main中的局部变量可以与函数power的形参x、n同名,但它们的作用域不同,只能作用于定义它的函数。

■ 当第1次执行函数调用power(x,n)时,把x的值(即1)传递给形参x,把n的值(即2)传递给形参n,在函数power内改变了形参n的值(由2变化为0),但并不能改变函数main中变量n的值。

3/27/2015

华中科技大学计算机学院 甘卓斌

48

5.3.2 参数的值传递

2/2

□ 传地址（引用）调用

- 传地址（引用）调用是将变量的地址传递给函数，函数既可以使用，也可以改变实参变量的值。
- 标准库函数scanf就是一个引用调用的例子，通过地址实参返回一个或多个数据。
- 程序员也可以定义这种带有地址形参的函数，这部分内容将在第9章中介绍。

3/27/2015

华中科技大学计算机学院 甘早斌

49

5.4 作用域与可见性

□ 5.4.1 概述

□ 5.4.2 局部变量和全局变量

□ 5.4.3 作用域规则*

□ 5.4.4 可见性

3/27/2015

华中科技大学计算机学院 甘早斌

50

5.4.1 概述

- 作用域和可见性是对一个问题的两种角度的思考。
- 作用域是指标识符（变量或函数）的有效范围，也就是指程序正文中可以使用该标识符的那部分程序段。
- 有局部和全局两种作用域：
 - 局部作用域表示只能在一定的范围内起作用，只能被一个程序块访问；
 - 全局作用域表示可以在整个程序的所有范围内起作用，可由程序中的部分或所有函数共享，程序块是带有说明的复合语句（包括函数体）。
 - 代码中的变量，按照作用域可分为局部变量和全局变量。函数都是全局的。

3/27/2015

华中科技大学计算机学院 甘早斌

51

5.4.2 局部变量和全局变量

□ 局部变量

- 在函数内部定义的变量，作用域是定义该变量的程序块，程序块是带有说明的复合语句（包括函数体）。不同函数可同名，同一函数内不同程序块可同名。形式参数是局部变量。

□ 外部变量

- 在函数外部定义的变量，其作用域从其定义处开始一直到其在文件的末尾，可由程序中的部分或所有函数共享。

3/27/2015

华中科技大学计算机学院 甘早斌

52

□ extern声明

- 在函数中使用外部变量，一般要对该变量进行引用性声明，说明它的类型。只有在函数内经过引用性声明的外部变量才能使用，外部变量的说明符为extern。但在一个外部变量定义之后的函数内使用可不再加以extern声明。
- 例5.5 改写例5.1，把magic定义为外部变量。这需要修改函数GuessNum的调用、原型和定义。
- [源程序\ex5_5.c](#)

3/27/2015

华中科技大学计算机学院 甘早斌

53

```

1. include <stdio.h>
2. #include <stdlib.h>
3. #include <time.h>
4. #define MAX_NUMBER 10
5. int GetNum(void); /* 函数原型 */
6. void GuessNum(void); /* 函数原型 */
7. int magic; /* 外部变量magic的定义 */
8. int main(void)
9. {
10.     char command;
11.     extern int magic; /* 外部变量magic的声明 */
12.     printf("This is a guessing game\n\n");
13.     srand(time(NULL)); /* 用系统时间初始化随机数生成器 */
14.     do {
15.         magic = GetNum(); /* 调用GetNum产生随机数供猜测 */
16.         GuessNum(); /* 调用GuessNum猜出这个数 */
17.         printf("Play again? (Y/N) ");
18.         scanf("%1s", &command); /* 询问是否继续 */
19.     } while (command == 'Y' || command == 'y');
20.     return 0;
21. }

```

```

1. 1. /*****
2. 2. 函数名称: GuessNum
3. 3. 函数功能: 游戏者根据提示反复猜数, 直至猜对为止, 输出猜数的次数。
4. 4. 函数参数: 无
5. 5. 函数返回值: 无   ***/
6. 6. void GuessNum ()
7. 7. {
8. 8.     int guess, counter = 0;
9. 9.     extern int magic;    /* 外部变量magic的声明 */
10. 10.    for (;;) {
11. 11.        printf("guess it: ");
12. 12.        scanf("%d", &guess);
13. 13.        counter++;        /* 统计猜的次数 */
14. 14.        if (guess == magic) { /* 猜对 */
15. 15.            printf("You guessed the number by %d times!\n", counter);
16. 16.            return;
17. 17.        }
18. 18.        else if (guess < magic) printf("Too low. Try again.\n"); /* 猜小了 */
19. 19.        else printf("Too high. Try again.\n"); /* 猜大了 */
20. 20.    }
21. 21. }

```

□ 程序分析

- magic是在函数外部定义的, 它是外部变量, 由于函数main和GuessNum使用了它, 所以在这两个函数中编写了一条extern声明语句, 该声明语句除了在前面加了关键字extern外, 其余与该外部变量的定义相同。
- 在该例中, 两条extern声明是多余的, 均可以省略, 因为magic的定义出现在函数main和GuessNum之前。

3/27/2015

华中科技大学计算机学院 甘早斌

56

□ 外部变量的定义性声明和引用性声明

- 局部变量只有定义性声明, 没有引用性声明。
- 而外部变量有定义性声明和引用性声明, 两者具有严格的区别。
- 外部变量的定义必须在所有的函数之外, 且只能定义一次。(Why?)
- 外部变量的引用性声明既可以出现在函数内, 也可以出现在函数外, 而且可以出现多次, 仅用于通报变量的类型, 并不分配存储单元, 只是表明在代码中要按声明的类型使用它。
- 外部变量的初始化只能出现在其定义中。

3/27/2015

华中科技大学计算机学院 甘早斌

57

□ 外部变量 vs 形式参数

- 函数之间的数据联系除了通过形参外, 还可通过外部变量。
 - 在C程序的不同源文件之间, 或者在同一源文件的不同函数之间必须共享变量时, 外部变量是很有用的。
- 从结构化程序设计的观点来看, 不要过于依赖外部变量! ??
 - 复用性差、函数的独立性降低
 - 可维护性差: 在程序维护期间, 一旦外部变量的改变 (如改变数据类型), 则所有调用该变量的函数都需要检查
- 一般来讲, 对于函数, 通过形参进行通信比通过外部变量通信更好。这样有助于提高函数的通用性, 减少副作用。

3/27/2015

华中科技大学计算机学院 甘早斌

58

5.4.2 作用域规则**

□ (1) 文件范围

- 其作用域开始于文件开头, 结束于文件结尾。
- 全局变量和函数具有文件作用域, 从定义它们的位置开始一直到本文件结束。
- 全局变量和函数如果在定义时使用了static存储类关键字, 它们的作用域只限定在本文件以内; 否则, 它们的作用域可以通过extern声明语句扩展到定义点之前及其他文件, 使得它们在定义之前可以使用, 以及其他与定义所在文件不同的源文件中的函数也可以使用。
- 关于static和extern的详细用法, 见5.5节。

3/27/2015

华中科技大学计算机学院 甘早斌

59

□ (2) 块范围

- 其作用域开始于左大花括号 “[”, 结束于右大花括号 “]”。
- 在复合语句内定义的变量其作用域属于块范围, 就在定义该变量的块内。在函数的开始部分定义的变量和函数的形式参数, 其作用域也属于块范围, 就在定义该变量的函数内部。
- 在不同函数中的同名变量相互之间没有任何关系。

□ (3) 函数原型范围

- 在函数原型中说明的变量只在函数原型内有效, 开始于原型左括号 “(”, 结束于原型右括号 “)”。

□ (4) 函数范围

- 开始于函数体的左大花括号 “[”, 结束于函数体的右大花括号 “]”。
- 只有goto语句的标号属于函数范围, 只能作用于同一函数内。

3/27/2015

华中科技大学计算机学院 甘早斌

60

□ 区分外部变量的引用性声明和定义性声明

■ 外部变量的引用性声明用于通报变量的性质（主要是变量的类型）

■ 外部变量的定义性声明则一方面通报变量性质；另一方面还产生存储空间分配。如果在函数外部有如下说明语句：

```
int sp; // 定义性声明，sp的类型为int，存储单元为2字节。
```

■ 而如下语句：

```
extern int sp; // 引用性声明，仅说明sp是一个类型为int的外部变量，系统不会为其分配存储单元。
```

■ 注意：

□ 在一个程序中对一个外部变量只能在某个文件中定义一次，而外部变量的引用性声明可以出现多次。

□ 外部变量的初始化只能出现在其定义中。

3/27/2015

华中科技大学计算机学院 甘卓斌

61

5.4.3 可见性

□ 作用域的嵌套

■ 程序块可以多重嵌套，每个块都可以定义自己的变量名。

■ 外层块的变量名在内层块中是有效的。

■ 如果一个变量名a同时出现在外层块和内层块中，外层a的作用域包含了内层a的作用域，这称为作用域的嵌套。

□ 当内层的变量和外层的变量同名时，在内层里，外层的变量暂时失去了可见性，是不可见的。

3/27/2015

华中科技大学计算机学院 甘卓斌

62

□ 块多重嵌套时变量的可见性

■ 外层块的a在内层块中不可见。在内层块中对a的修改不会影响到外层块的a。在进入复合语句之前和退出复合语句之后输出的a均为2。而变量b在内层块没有重定义，它在整个函数体都可见。

■ 同样地，全局变量和局部变量也可以同名。在局部变量的作用域内，全局变量不可见。

```
< int a=2,b=4;          /* 外层a的作用域 */
  printf("a=%d,b=%d\n"); /* 输出 a=2,b=4 */
< int a;              /* 内层a的作用域 */
  a=3;b=5;
  printf("a=%d,b=%d\n",a,b); /* 输出 a=3,b=5 */
> printf("a=%d,b=%d\n",a,b); /* 输出 a=2,b=5 */
>
```

3/27/2015

华中科技大学计算机学院 甘卓斌

63

5.5 存储类型

□ 5.5.1 变量和函数的属性

□ 5.5.2 存储类型auto

□ 5.5.3 存储类型extern

□ 5.5.4 存储类型static

□ 5.5.5 存储类型register

□ 5.5.6 新增存储类型_Thread_local**

3/27/2015

华中科技大学计算机学院 甘卓斌

64

5.5.1 变量和函数的属性

□ 变量和函数都有两个属性：数据类型和存储类型。

■ 函数的存储类型决定函数的作用域，可使用的关键字有extern和static。

■ 变量的存储类型决定变量的作用域、存储分配方式、生命周期和初始化方式，使用关键字：auto、extern、static和register。

■ 存储分配方式

□ 在何时、何处给变量分配存储单元；

■ 生命周期

□ 变量在内存中的存在期；

■ 初始化方式

□ 在定义变量时如果没有显示初始化，是否有缺省初值，如果有显示初始化，赋初值操作如何执行（执行一次还是执行多次）。

3/27/2015

华中科技大学计算机学院 甘卓斌

65

5.5.2 存储类型auto

□ 局部变量的缺省存储类型是auto，称自动变量

```
auto int a; /* 等价于int a; 也等价于auto a; */
```

□ 作用域

■ 局部于定义它的块，从块内定义之后直到该块结束有效。

□ 存储分配方式

■ 动态分配方式，即在程序运行过程中分配和回收存储单元。

□ 生命周期

■ 短暂的，只存在于该块的执行期间。

□ 初始化方式

■ 定义时没有显示初始化，其初值是不确定的；有显示初始化，则每次进入块时都要执行一次赋初值操作。

3/27/2015

华中科技大学计算机学院 甘卓斌

66

5.5.3 存储类型extern

□ 1. 外部变量

- 存储类型是extern, 在定义时不使用关键字extern。
- 作用域
 - 从定义之后直到该源文件结束的所有函数, 通过用extern做声明, 外部变量的作用域可以**扩大到整个程序的所有文件**。
- 存储分配方式
 - 静态分配方式, 即程序运行之前, 系统就为外部变量在静态区分配存储单元, 整个程序运行结束后所占用的存储单元才被收回。
- 生命周期
 - 永久的, 存在于整个程序的执行期间。
- 初始化方式
 - 定义时**没有显示初始化, 初值0**; 有显示初始化, 只执行一次赋值初值操作, 且初始化表达式必须是一个常量表达式。

3/27/2015

华中科技大学计算机学院 甘早斌

67

□ 例5.6 关键字extern的用法 见教材P119

- 修改例5.5, 将整个程序放在两个文件ex5_6_1.c和ex5_6_2.c中
 - 源程序ex5_6_1.c
 - 源程序ex5_6_2.c
 - 当编写大型程序时, 构造多个文件是很重要的, 每个文件可以单独编译。
- 程序分析
- 该程序包含2个源文件ex5_6_1.c和ex5_6_2.c, 外部变量magic在ex5_6_1.c文件中定义, 在ex5_6_2.c文件中必须声明后才能使用它。ex5_6_2.c文件中的第4行代码就是外部变量magic的声明, extern告诉编译器变量magic被定义在别处, 可能在本文中, 也可能在另一个文件中, 但编译器并不知道它在哪个文件中定义, 因此让连接程序查找。
 - 如果连接程序找到了外部变量magic的正确定义, 它就会指明其位置, 从而解决对该变量的引用。如果连接程序没有找到magic的定义, 它就会发出错误信息并且不生成可执行文件。

3/27/2015

华中科技大学计算机学院 甘早斌

68

□ 2. 外部函数

- 函数一般是全局的, 作用域属于文件范围, 对程序的任何部分都是可见的, 其默认存储类型是extern, 这种函数称为外部函数。
- 在函数定义和函数原型中都可以使用关键字extern。例如:
 - extern double power(int, int);
 - 是函数power的函数原型, 其函数定义可为:
 - extern double power(int x, int n) {...}
 - power是外部函数, 它可被本文件和其他文件中的函数调用。
- 函数定义时一般省略extern, 在其他需要调用它的文件中, 一般用extern声明。例如, 例5.6的ex5_6_1.c文件中的函数原型常写作:
 - extern int GetNum(void);
 - extern void GuessNum(void);

3/27/2015

华中科技大学计算机学院 甘早斌

69

5.5.4 存储类型static

□ 关键字static有两个重要而截然不同的用法:

- (1) 用于定义局部变量, 称为静态局部变量。
- (2) 用于定义外部变量, 称为静态外部变量。与外部变量区别在于作用域不同。

□ 静态变量的特点

- 存储分配方式: 静态分配方式。
- 生命周期是永久的, 存在于整个程序的执行期间。
- 缺省初值为0, 初始化表达式必须是一个常量表达式。

3/27/2015

华中科技大学计算机学院 甘早斌

70

□ 1. 静态局部变量

- 静态局部变量的作用域和自动变量一样, 只作用于定义它的块。
- 由于静态局部变量在**程序执行期间不会消失**, 因此, 它的值有**连续性**, 当退出块时, 它的值能保存下来, 以便再次进入块时使用。而**自动变量的值在退出块时都丢失了**。
- 如果定义时静态局部变量有显示初始化, 只执行一次赋值初值操作, 而自动变量每次进入时都要执行赋值初值操作。

3/27/2015

华中科技大学计算机学院 甘早斌

71

□ 例5.7 编程计算 $1!+2!+3!+4!+\dots+n!$

- 将求阶乘定义成函数, 由于 $n! = n * (n-1)!$, 可以直接利用上次调用后的值算出, 使**计算量最小**。为了使局部变量的值在离开函数后能保存, 必须在定义时加static, 成为静态局部变量。
- 源程序ex5_7.c

3/27/2015

华中科技大学计算机学院 甘早斌

72

```

1. #include<stdio.h>
2. long fac(int); /* 函数原型 */
3. int main(void)
4. {
5.     int i,n;
6.     long sum=0;
7.     printf("input n(n>0):\n");
8.     scanf("%d",&n);
9.     for(i=1;i<=n;i++)
10.         sum+=fac(i);
11.     printf("1!+2!+...+%d!=%ld\n",n,sum);
12.     return 0;
13. }

14. long fac(int n)
15. {
16.     static long f=1; /* 静态局部变量 */
17.     f*=n;
18.     return f;
19. }

```

3/27/2015

华中科技大学计算机学院 甘早斌

73

□ 程序分析

- 在函数fac内的变量f是静态局部变量。在第一次调用函数fac时，把f初始化为1，在退出函数时，f的值被保存在内存中。当再次调用函数时，就不会再对f进行初始化，而使用上次退出函数时保存在内存中的值。虽然无论使用与否，f都占据内存，但是，变量f只能被函数fac访问，其他函数不能访问。
- 请读者思考，如果将f定义为自动变量（即去掉关键字static），程序的输出结果如何？
- 使用静态局部变量是为了多次调用同一函数时使变量能保持上次调用结束时的结果。即**静态局部变量的值有记忆性**。

3/27/2015

华中科技大学计算机学院 甘早斌

74

□ 2. 静态外部变量

- 静态外部变量和外部变量的区别是作用域的限制。
- 静态外部变量只能作用于定义它的文件，其它文件中的函数不能使用。
- 外部变量用extern声明后可以作用于其它文件。
- 使用静态外部变量的好处在于：当多人分别编写一个程序的不同文件时，可以按照需要命名变量而不必考虑是否会与其它文件中的变量同名，保证文件的独立性。
- 和局部变量能够屏蔽同名的外部变量一样，一个文件中的静态外部变量能够屏蔽其他文件中同名的外部变量。在静态外部变量所在的文件中，同名的外部变量不可见。

3/27/2015

华中科技大学计算机学院 甘早斌

75

□ 例5.8 伪随机数发生器的实现与使用

- 产生随机序列需要给定初始种子，因此种子seed是各随机数发生器函数所共享的变量，应定义在函数外。而且，seed只提供給random、random和probability等产生随机数的函数操作，并不希望任何其他函数访问它们，因此将函数srandom、random和probability，以及它们所操作的种子seed设计在一个源文件ex5_8_1.c中，在定义seed时加上static，限制其作用域只在本文件内。将函数main设计在另一个源文件ex5_8_2.c中。

□ 源程序\ex5_8_1.c

□ 源程序\ex5_8_2.c

3/27/2015

华中科技大学计算机学院 甘早斌

76

□ 程序分析

- 以上程序由两个文件组成，在文件ex5_8_1.c中定义了一个静态外部变量seed，依赖变量seed的旧值，调用函数random和probability为变量seed产生一个新值。
- 由于seed是静态外部变量，它对本文件的这些函数来说是共享的，它的值在函数调用之间被保存下来了，但是，它对本文件来说是私有的，在文件ex5_8_2.c中不能用extern对seed做声明，也不能在main函数中使用seed。现在可以在文件ex5_8_2.c中调用这些随机数发生器函数而不必担心副作用。

3/27/2015

华中科技大学计算机学院 甘早斌

77

□ 3. 静态函数

- 如果要把函数的作用域限制在定义它的文件中，在函数定义时必须使用关键字static，这种函数称为静态函数。例如：
 - ① double power(int,int); /* power函数原型 */
 - ② void main(void)
 - ③ {
 - ④ ... /* 函数power 可被调用，但在其他文件不能被调用 */
 - ⑤ }
 - ⑥ static double power(int x,int n); /* 静态函数 */
 - ⑦ {
 - ⑧ ...
 - ⑨ }
- 和静态外部变量一样，静态函数也只作用于所在文件，不同文件中的静态函数可以同名，保证文件的独立性。

3/27/2015

华中科技大学计算机学院 甘早斌

78

5.5.5 存储类型register

- 用来定义局部变量，register建议编译器把该变量存储在计算机的高速硬件寄存器中，除此之外，其余特性和自动变量完全相同。
- 使用register的目的是为了提高程序的执行速度。程序中最频繁访问的变量，可声明为register。
 - ① register int i; /* 等价于register i; */
 - ② for (i=0;i<=N;i++) { ... }
- 不可多
- 必要时使用。

3/27/2015

华中科技大学计算机学院 甘卓斌

79

** 5.5.6 新增存储类型_Thread_local

- C11增加了_Thread_local存储类关键字以支持多线程。
- 进程和线程
 - 进程和线程都是操作系统的概念。进程是应用程序的执行实例，线程是进程内部的一个执行单元。系统创建好进程后，实际上就启动了该进程的主执行线程，主执行线程以函数（比如main）地址形式，将程序的启动点提供给Windows系统。主执行线程终止了，进程也就随之终止。
 - 每一个进程至少有一个主执行线程，它无需由用户去主动创建，是由系统自动创建的。用户根据需要在应用程序中创建其他线程，多个线程并发地运行于同一个进程中。一个进程中的所有线程都在该进程的虚拟地址空间中，共同使用这些虚拟地址空间、全局变量和系统资源，所以线程间的通信非常方便。

3/27/2015

华中科技大学计算机学院 甘卓斌

80

□ 线程局部存储技术

- 线程局部存储（ThreadLocal）技术是多线程技术中用于解决并发问题的技术。其原理是将一块内存与线程关联，每个线程访问的变量都存在于本线程的局部存储区中，因此多个线程间访问相同的变量名时不会产生并发问题。关键字_Thread_local就是C语言用来实现线程局部存储的。
- 使用_Thread_local关键字声明线程（thread）变量，可以很好地解决变量并发访问的冲突问题，它为每个使用该变量的线程提供单独的变量副本，并且在线程运行之前初始化。所以每一个线程都可以独立地改变自己的副本，而不会影响到其他线程所对应的副本

3/27/2015

华中科技大学计算机学院 甘卓斌

81

□ _Thread_local关键字

- 下面的代码声明了一个整数线程局部变量，并用一个值对其进行初始化：


```
_Thread_local int tls_i = 1;
```
- 只能对外部变量和静态变量（包括函数内定义的静态变量）指定_Thread_local修饰符，不可以用它声明自动变量。
- _Thread_local属性只能应用于变量声明和定义，不能用于函数声明或定义。例如，以下代码将生成一个编译器错误：


```
_Thread_local void func();        /* 错误 */
```
- 线程本地对象的声明和定义必须全部指定_Thread_local属性，例如：


```
□ _Thread_local extern int tls_i;        /* 声明 */
□ _Thread_local int tls_i;                /* 定义 */
```
- C11还引进了一个标准C库的头文件threads.h，里面提供了宏、类型及支持多线程的函数。

3/27/2015

华中科技大学计算机学院 甘卓斌

82

本章小结

- 结构化编程是一种对复杂问题“分而治之”的策略，也是一种尽量简化控制流和使用自顶向下设计思想的编程方法，自顶向下设计思想就是把一个问题按功能分解为若干模块。
- C语言中提供的函数机制可使程序模块化。C程序的整体由一个或多个函数组成，每个函数都具有各自独立的功能，这些函数可以分类存放在多个文件中。因此，函数是C语言中最重要的概念之一。
- 用C语言设计程序，任务就是编写函数。通过编写函数定义，创建一个由函数头和函数体组成的函数。函数定义中的形参属于局部变量，作用域限定在函数体内。函数调用时的实参是具有确定值的表达式。在调用函数时，实参的值单向传递给形参，这意味着当把一个变量作为参数传递给函数时，它的值在调用函数中保持不变。函数原型声明了函数返回值的类型，以及参数的个数、类型和顺序，编译器用函数原型校验函数调用，将实参转换为原型相应参数的类型，保证函数调用的正确性。

3/27/2015

华中科技大学计算机学院 甘卓斌

83

- 变量的作用域和生命周期要特别注意。局部变量只在其定义的块内有效，局部变量默认的存储类型是auto，其生命周期是短暂的，只存在于定义所在块的执行期间；局部变量若被声明为static，则其生命周期是永久的，存在于程序的整个执行期间。
- 外部变量的生命周期是永久的，其作用域可以是整个程序，但在定义点之前或其他文件中使用，必须用extern声明。
- 外部变量若被声明为static，其作用域限制在定义所在的文件，其他方面的存储特性同外部变量。寄存器变量除在可能的情况下用硬件寄存器分配存储外，其他方面与自动变量完全相同。

3/27/2015

华中科技大学计算机学院 甘卓斌

84

Assignments:

- 必做题:
 - 5.1, 5.2, 5.3, 5.4, 5.5, 5.6, 5.7, 5.8, 5.9, 5.10, 5.11, 5.12, 5.13
- 我建议:
 - 后面习题, 每题都做, 并且搞懂!

3/27/2015

华中科技大学计算机学院 甘卓斌

85