







5.1.1 结构化程序设计 1/8

display 1/8

stantal 1/8

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

### 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普退休教授,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。

华中科技大学计算机学院 甘早斌

3/27/2015

# 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。

5.1.1 结构化程序设计 4/8

■ 基本结构

■ 顺序结构

■ 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、双选择和多选择三种形式。

■ 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)即才可终止循环。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。

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。

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

### 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 便干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 牛许多问题 ■ 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 甘早斌 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

□ 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局,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□ 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

□【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决数。
■ 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。

□ 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的随机数供游戏者精测;
□ (2) 游戏者精数,直至精对。





□ 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、函数如能。函数多数。函数返回值等内容
■ 如有必要还可搅用作者。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。

□ int \_\_cdecl \_\_MINGW\_NOTHROW rand (void);
■ rand()是stdiib.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。

□ 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

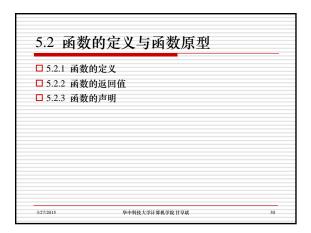
□ 函数 void GuessNum(int x)

■ 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

```
□ 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rand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
```

# 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。

□ 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。



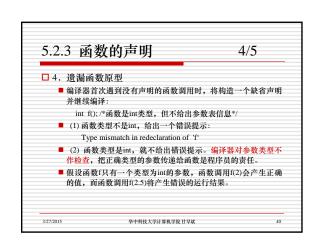














□ 5.3.1 函数诉	
□ 5.3.2 参数的	





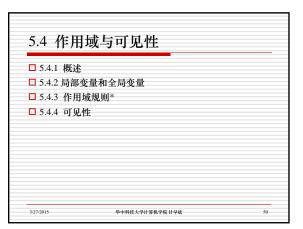




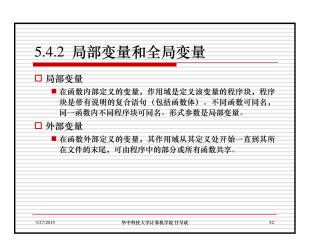
### 5.3.1 函数调用 4/4 □ 3. 实参的求值顺序 ■ 函数调用中实参之间的逗号是分隔符,它与顺序求职运算符',' 不同 ■ 实参的求值顺序由具体实现确定,有的按从左至右的顺序计算, 有的按从右至左的顺序计算 a=1: power(a,a++) ----从左至右: power(1,1) ----从右至左: power(2,1)(多数) ■ 为了保证程序清晰、可移植,应避免使用会引起副作用的实 参表达式. 3/27/2015 华中科技大学计算机学院甘早斌 47

# 5.3.2 参数的值传递 □ 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。 ■ <u>源程序ex5 4.c</u> ■ 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or的循环次数,这样就不再需要引入局部变量;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。





### 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。



```
□ exterm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 改写例形式,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。
■ <u>源程序\ex5</u> 5.c
```

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

```
函数名称: GuessNum
    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。
    函数参数: 无
   -
函数返回值: 无 ***/
    void GuessNum ( )
         int guess,counter = 0;
         extern int magic;
                            /* 外部变量magic的声明 */
10.
         for (;;) {
               printf("guess it: ");
               printi( guess ... ,
scanf("%d", &guess);
counter++; /* 统计猜的次数 */
12.
14.
                if (guess == magic) { /* 猜对 */
15.
                   printf("You guessed the number by %d times!\n\n", counter);
17.
18.
                 else if (guess < magic) printf("Too low. Try again.\n");  /* 猜小了 */
19.
                  else printf("Too high. Try again.\n");
                                                         /* 猜大了 */
20.
         }
```

## □ 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而加了关键字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。

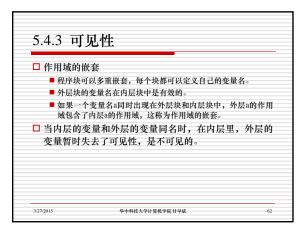
□ 外部变量的定义性声明和引用性声明
■ 局部变量只有定义性声明,没有引用性声明。
■ 而外部变量有定义性声明和引用性声明,两者具有严格的区别。
■ 外部变量的定义必须在所有的函数之外,且只能定义一次。(Why')
■ 外部变量的引用性声明既可以出现在函数内,也可以出现在函数外,而且可以出现多次,仅用于通报变量的类型,并不分配存储单元,只是表明在代码中要按声明的类型使用它。
■ 外部变量的初始化只能出现在实定义中。

□ 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 必須共享变量时,外那变量是很有用的。
■ 从结构化程序设计的观点来看,不要过于依赖外部变量! ??□ 复用性慈、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。

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extem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。

□ (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形式参数,其作用域也属于块范围,或在定义该变量的函数内径何关系。
■ 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。





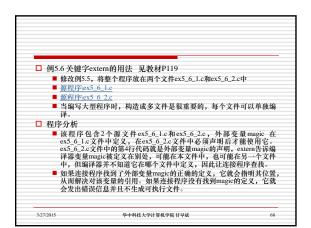




## 5.5.1 变量和函数的属性 ② 变量和函数都有两个属性:数据类型和存储类型。 ■ 函数的存储类型决定函数的作用域,可使用的关键字有extern和static。 ■ 变量的存储类型决定变量的作用域、存储分配方式、生命周期和初始化方式,使用关键字: auto、extern、static和register。 ■ 存储分配方式 □ 在何时、何处给变量分配存储单元; ■ 生命周期 □ 变量在内存中的存在期: ■ 初始化方式 □ 在定义变量时如果未显示初始化,是否有缺省初值,如果有显示初始化,赋初值操作如何执行(执行一次还是执行多次)。

# 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。

### 







□ 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,只执行一次赋初值操作,而自动变量每次进入时都要执行赋初值操作。

□ 例 5.7 编程计算1!+2!+3!+4!+...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,使计算量量小。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 lex5 7.c

```
#include<stdio.h>
                       /* 函数原型 */
    long fac(int);
     int main(void)
      int in:
      long sum=0;
      printf("input n(n>0):\n"); scanf("%d",&n);
      for(i=1:i<=n:i++)
       sum+=fac(i);
 11. \quad printf("1!+2!+..+\%d!=\%ld\n",n,sum); \\
      return 0;
13. }
14. \ long \ fac(int \ n)
    static long f=1; /* 静态局部变量 */
f *=n;
16.
18.
     return f:
19. }
3/27/2015
                             华中科技大学计算机学院 甘早斌
                                                                                 73
```



□ 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时名的外部变量一样。

□ 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只提供给srandom、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

□ 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. 静态函数
■ 如果要担函数的作用城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.nt): /\* 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 ....
⑤ }
■ 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。

### 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。

华中科技大学计算机学院 甘早斌

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 甘早斌

### □ 线程局部存储技术 ■ 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,而不会影响其他线程所对应的副本

### 本章小结

3/27/2015

- 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。函数调用时的实多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 甘早斌

- □ 变量的作用域和生命期要特别注意。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 甘早斌

14

84

