C语言

第4讲：C语言的控制结构

赵岩

哈尔滨工业大学软件学院

August 20, 2011

目录

目录

目录

目录

数据结构和算法

- 程序=数据结构+算法
- 数据结构
  - 数据结构是计算机存储、组织数据的方式
  - 实例：链表，树，图等数据结构
- 算法
  - 为解决一个具体问题而采取的确定的有限的操作步骤
  - 实例
    - 1+2+...+100
    - 最大公约数

最大公约数算法

- 求1000005和1000000的公约数
- 最大公约数算法:
  1. x对y求余数r
  2. r=0，y为最大公约数
  3. r!=0, x=y,y=r,跳到第一步
- 衡量算法的复杂度
- 好的算法可以**极大**地提高你的效率。
  - $2^n \rightharpoonup n^2 \rightharpoonup log(n) \rightharpoonup M$

算法的特性

- 有穷性
  - 在合理的时间内完成
- 确定性，无歧义
  - 如果x≥0，则输出Yes；如果x≤0，则输出No；
- 有效性
  - 能有效执行
  - 负数开平方
- 没有输入或有多个输入
- 有一个或多个输出

算法的描述方法

- 自然语言描述
- 流程图
- NS流程图
- 伪码（我个人用的最多）

算法的学习方法

- **记住**常用的代码片段，code snipet
  - 基本输入输出
  - 数组遍历
  - 文件打开
  - .....
- **熟练**应用三种基本结构及应用场景
- **掌握**简单的数据结构和算法
  - 链表
  - 查找和排序，随机数

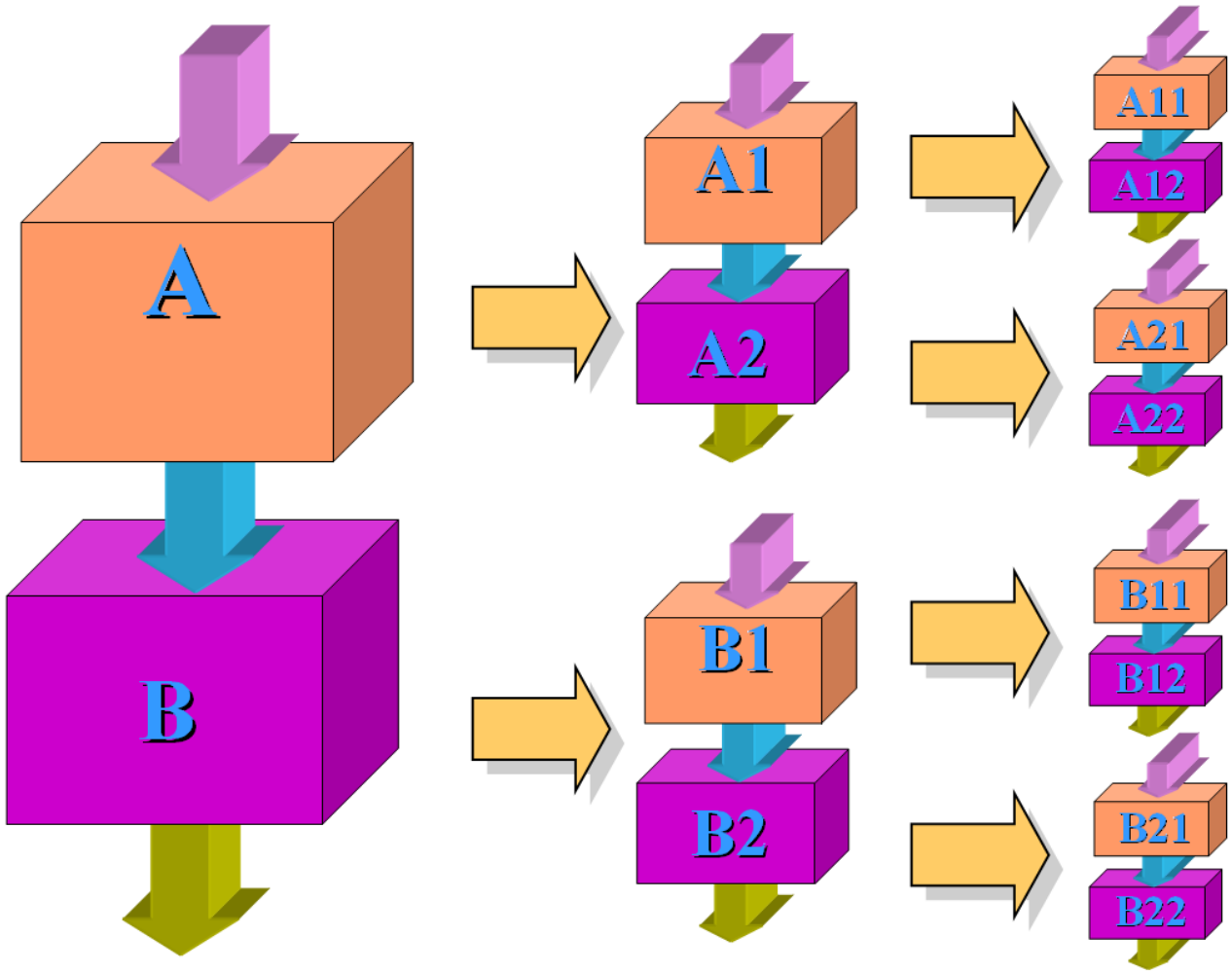
算法参考书

- **葵花宝典：**The Art of Computer Programming
- **独孤九剑：**Introduction to Algorithms **《算法导论》高等教育出版社**
- **九阴真经：**Computer Algorithms: Introduction to Design and Analysis

结构化的设计核心

- 采用**顺序**、**选择**和**循环**三种基本结构
  - 只有一个入口
  - 只有一个出口
  - 无死语句，即不存在永远都执行不到的语句
  - 无死循环，即不存在永远都执行不完的循环
- 采用“自顶向下、逐步求精”和模块化的方法进行结构化程序设计
  - 先全局后局部
  - 先整体后细节
  - 先抽象后具体

结构化的设计方法

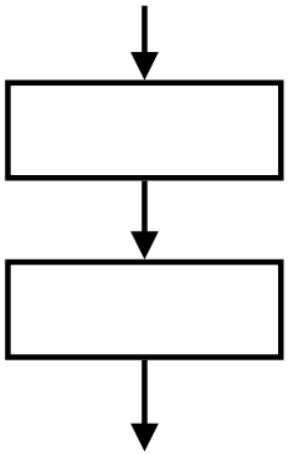


C程序的三种基本结构

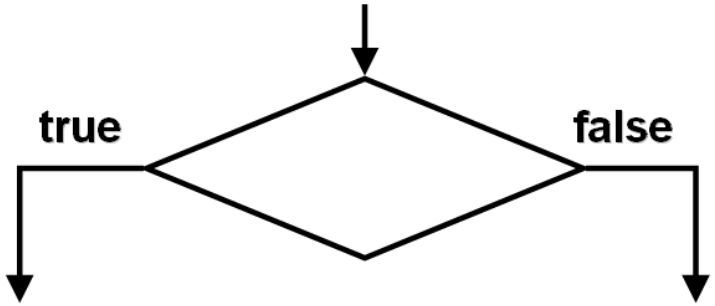
- 顺序结构、选择结构、循环结构
- 已经证明，任何程序均可只用这三种结构实现
- 只用这三种结构的程序，叫结构化程序
- 程序“必须”符合结构化规则

三种基本结构流程图

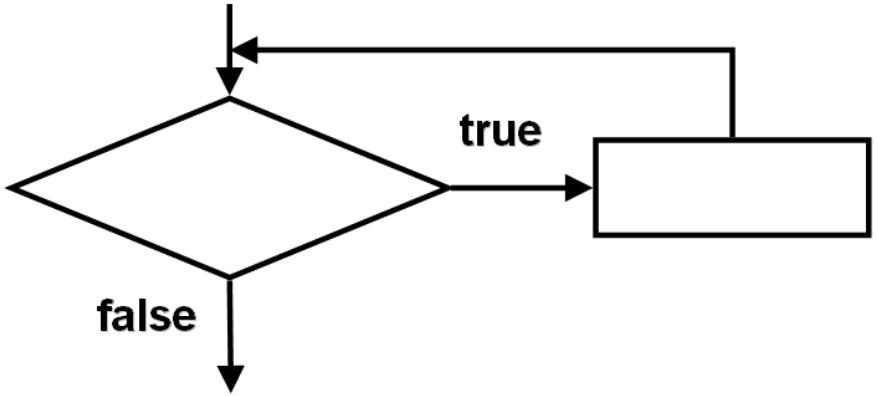
顺序结构



选择结构



循环结构



语句块

- {}括住的若干条语句构成一个语句块
- 语句块内可以定义变量
  - 变量必须在语句块的开头定义
  - 变量仅在定义它的语句块内(包括下层语句块)有效
  - 同一个语句块内的变量不可同名，不同语句块可同名
    - 各司其职、下层优先
    - 避免在下层语句块内定义同名变量。
- 语句块通常用在判断和循环语句。

语句块实例

- 语句块中的变量只在定义它的语句块中有效
- **避免写出下面的程序**

Demo 1: local variable

```
1 main()
2 {
3       int   a = 0;
4       {
5           int   a = 1;
6           printf("Inside: a = %d\n", a);
7       }
8       printf("Outside: a = %d\n", a);
9 }
```

顺序结构程序实例

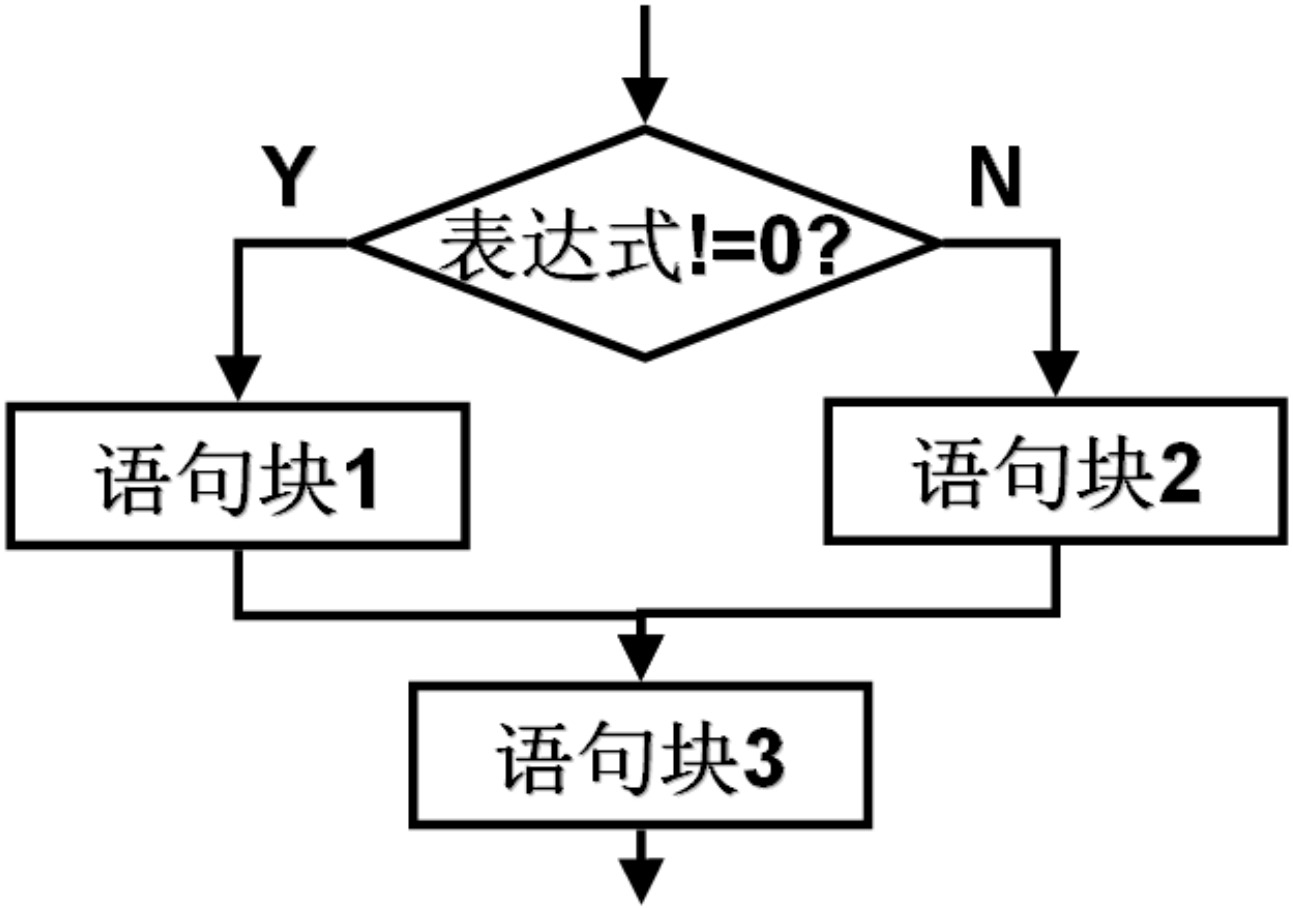
Demo 2: sequence structure

```
1  int x,b0,b1,b2;
2  scanf("%d",x);
3  b2 = x/100;
4  b1 = (x-b2*100)/10;
5  b0 = x%10;
6  printf(" %4d  %4d  %4d",b2,b1,b0);
```

问题

$2^{1000000}$有多少位？

if-else代码



if-else定义

- 选择结构的一种最常用形式
- **表达式**值非0（真）时，执行**语句块1**，**表达式**值为0（假）时，执行**语句块2**，然后执行**后续语句**
- else部分可以没有。

```
1  if (表达式){
2        语句块1;
3  }
4  else{
5        语句块2;
6  }
7        后续语句
```

else的配套问题

- x等于零的时候
  - y等于零,打印both zero
- x不等于零的时候
  - z=y/x

```
1       if(x==0)
2           if(y==0) printf("both zero");
3       else
4       {
5           z=y/x;
6       }
```

else的配套问题的实际含义

- else与最近的if配套

```
1  if(x==0)
2  {
3      if(y==0) printf("both zero");
4      else
5      {
6          z=y/x;
7      }
8  }
```

else的配套问题的解决办法

- 利用大括号封装第一个if

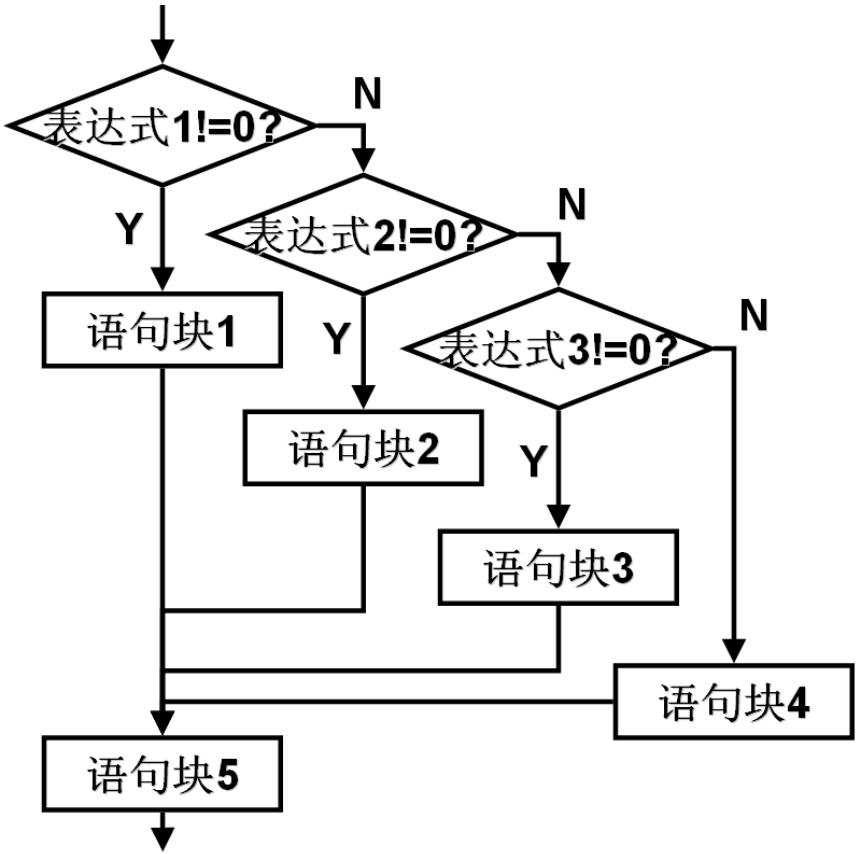
```
1  if(x==0)
2  {
3      if(y==0) printf("both zero");
4  }
5  else
6  {
7      z=y/x;
8  }
```

if-判断条件简写

- 非零值为真
- 零值为假

```
1        if(x)
2
3        if(x!=0)
```

多重选择




```
1  if (表达式1){
2      语句块1;
3  }
4  else if (表达式2){
5      语句块2;
6  }
7  else if (表达式3){
8      语句块3;
9  }
10 else{
11     语句块4;
12 }
13 后续语句;
```

多重选择实例

- 按“体指数”对肥胖程度进行划分：
  - 体指数t = w / h2 （体重w单位为公斤，身高h单位为米）
  - 当t < 18时，为低体重；
  - 当18 ⩽t < 25时，为正常体重；
  - 当25 < t < 27时，为超重体重；
  - 当t ⩾27时，为肥胖。

switch

```
1  switch (表达式)
2  {
3      case 整型常数1:
4          语句1;
5      case 整型常数2:
6          语句2;
7      default:
8          语句3;
9  }
```

- default可以没有，但最好不要省略

忽略break的问题

Demo 1: break problem

```
1  int a;
2  scanf("%d",&a);
3  switch (a){
4       case 1:
5            printf("you input 1\n");
6       case 2:
7            printf("you input 2\n");
8       case 3:
9            printf("you input 3\n");
10      default:
11           ;
12 }
```

switch实例

- 编程设计一个简单的计算器程序
  - 要求根据用户从键盘输入如下形式的表达式：操作数1 运算符op 操作数2
  - 然后，计算并输出表达式的值
  - 指定的运算符为
    - 加（+）
    - 减（-）
    - 乘（*）
    - 除（/）

switch实例代码

```
1  switch (c){
2      case '\n'
3      /* I skip break here */
4      case '-'
5          sub();
6          break;
7      case '+':
8          add();
9          break;
10     default:
11         break;
12         ;
13 }
```

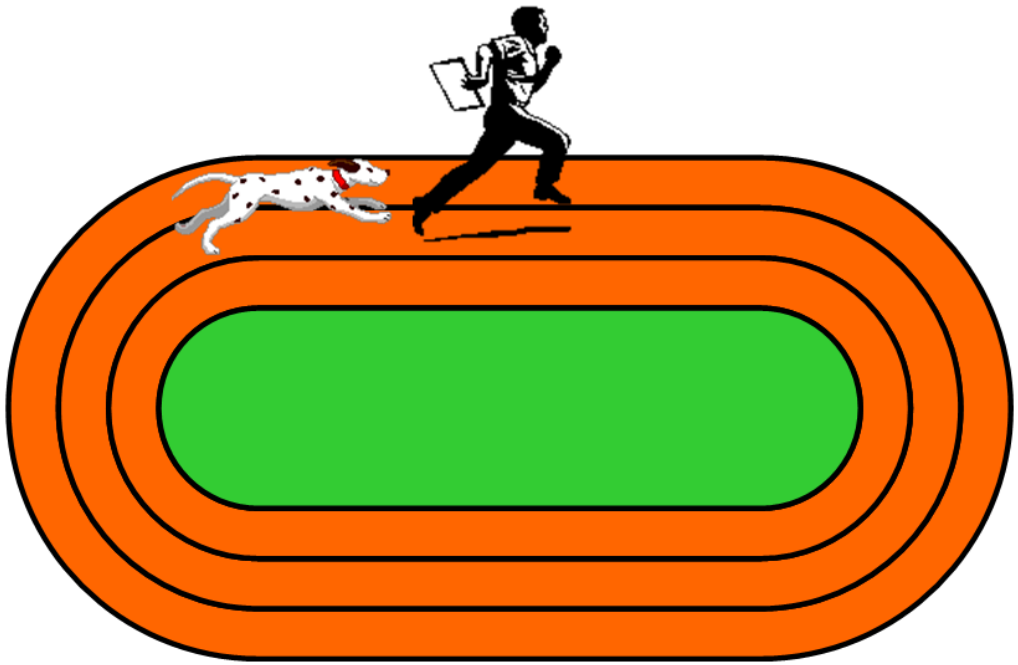
switch与if比较

- switch比else-if更清晰
- else-if比switch的条件控制更强大一些
  - else-if可以依照各种逻辑运算的结果进行流程控制
  - switch只能进行整数的==判断，条件表达式经过适当的计算变成整数。

```
1 mark = score / 10;
2 switch(mark){
3     case 9:
4     printf("80-90");
5     break;
6     ......
7 }
```

循环结构

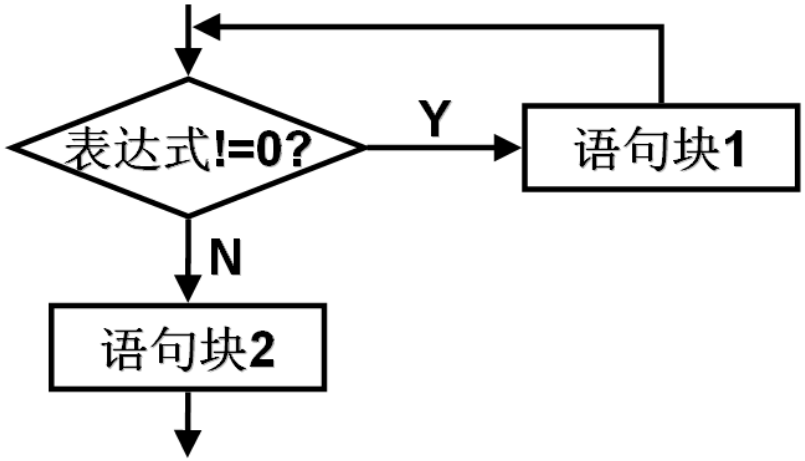
- “当型”循环
  - while循环
  - for循环
- “直到型”循环
  - do-while循环



while循环

- 只要表达式的值为真（非零），就重复执行语句块1，直到表达式值为0时止，开始执行后续语句

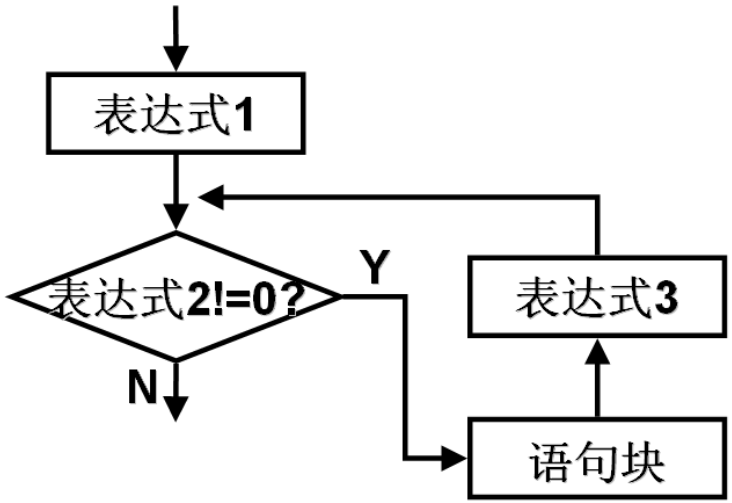
```
1  while (表达式){
2        语句块1
3  }
4  后续语句
```



for循环

```
1  for (表达式1; 表达式2; 表达式3){
2        语句块;
3  }
```

- 首先执行表达式1。如果表达式2的值为真（非零），就重复执行语句块和表达式3，直到表达式2的值为假（零）时止



for循环

- for的复杂表达式
  - 分号不可省略
  - 表达式2省略，恒为真
  - 表达式1和表达式3可以是用逗号分隔的
- for循环体内避免修改控制循环次数的循环变量。

```
1  for(i = 0,j = 0;;i++,j++)
```

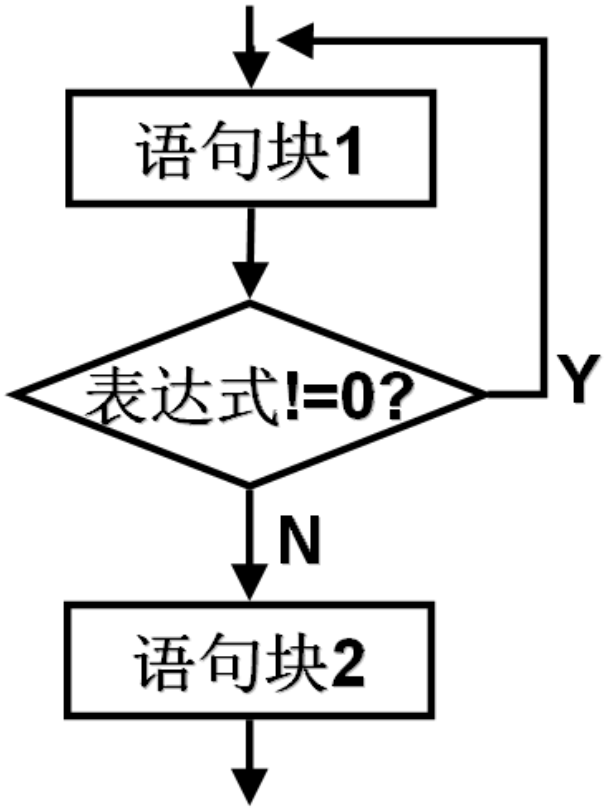
问题

for(i = 4,j = 0;i>j;i++,j=j+2) 循环几次？

do-while循环

```
1        do {
2             语句块1;
3        }
4        while (表达式);
5        后续语句;
```

- 首先执行语句块1，然后判断表达式的值。如果为假（零），继续向下执行，否则，再次执行语句块1。
- 与while的区别为语句块1会被执行至少一次



三种循环语句区别

- 如果循环次数已知，用`for`
- 如果循环次数未知，或者无初始化操作，用`while`
- 如果循环体至少要执行一次，用`do-while`
- **只是思路，不是定律**
- 如果能用一种循环完成，那么一定能用另外两种循环来完成。

判断，循环语句中的分号

- 在if、for和while语句之后一搬不加分号
  - 分号是一条空语句，它成为了循环体
- do...while语句之后一定有分号

```
1  while (i < 100); i++;
2  for (i = 0; i < 100; i++);
3  printf("%d", i);
4  /*
5  for (i = 0; i < 100; i++){
6      ;
7  }
8  printf("%d", i);
9  */
```

死循环

- 尽量避免死循环，它们使循环的中止条件变得不明朗。
  - 绝大多数程序不需要死循环。如果出现，往往都是bug
  - 时间过长的循环会造成“假死”现象，也要考虑解决

```
1 for (;;) {}
2 while (1) {}
3 do {} while (1);
```

死循环实例

```
1  for (i = 0;i<100;i++){
2        i=i-1;
3  }
```

```
1  for (i = 0;i<10000;i++){
2        if((i%10)==0) printf("%d",i);
3        for(j = 0;j<10000;j++){
4              for(k = 0;k<10000;k++){
5                    sum=i+j+k;
6              }
7        }
8  }
```

猜数游戏

- 先由计算机“想”一个1到100之间的数请人猜，如果人猜对了，则结束游戏，否则计算机给出提示，告诉人所猜的数是太大还是太小，直到人猜对为止。计算机记录人猜的次数，以此来反映猜数者“猜”的水平。

猜数游戏用到的随机函数

- 随机函数rand()
- #include <stdlib.h>
  - RAND_MAX在stdlib.h中定义，不大于双字节整数的最大值32767
- 产生[0,RAND_MAX] 之间的随机数
  - magic = rand();
- 产生[0,b-1] 之间的随机数
  - magic = rand() % b;
- 产生[a,a+b-1] 之间的随机数
  - magic = rand() % b + a;

随机函数内部实现

```
1  unsigned long int next = 1;
2  int rand(void)
3  {
4     next = next * 1103515245 + 12345;
5     return (unsigned int)(next/65536) % 32768;
6  }
7
8  void srand(unsigned int seed)
9  {
10    next = seed;
11 }
```

随机数种子

- 随机函数srand
  - 为函数rand()设置随机数种子来实现对函数rand所产生的伪随机数的“随机化”
- 使用计算机读取其时钟值并把该值自动设置为随机数种子，产生[1,100]之间的随机数
- 函数time()返回以秒计算的当前时间值，该值被转换为无符号整数并用作随机数发生器的种子

```
1 #include   <time.h>
2 srand(time(NULL));
3 magic = rand() % 100 + 1;
```

编程实例

- 古印度舍罕王决定赏赐他的的宰相达依尔，这位聪明的宰相指着8×8共64格的象棋盘说：陛下，请您赏给我一些麦子吧，就在棋盘的第一个格子中放1粒，第2格中放2粒，第3格放4粒，以后每一格都比前一格增加一倍，依此放完棋盘上的64个格子，我就感恩不尽了。
- 试编程计算舍罕王共要多少麦子赏赐他的宰相，这些麦子合多少立方米？（已知1立方米麦子约1.42e8粒）
- 总粒数为：$sum = 1 + 2^1 + 2^2 + 2^3 + \ldots + 2^{63}$

实例源码

```
1 #define CONST  1.42e8
2 int      n;
3 double term, sum = 0;   /*求和变量赋初值*/
4 for (n=1; n<=64; n++){
5       term = pow(2, n-1);
6       sum += term;          /*作累加运算*/
7 }
8 printf("sum = %e\n", sum);   /*总麦粒数*/
9 printf("volum=%e\n", sum/CONST);  /*体积数*/
```

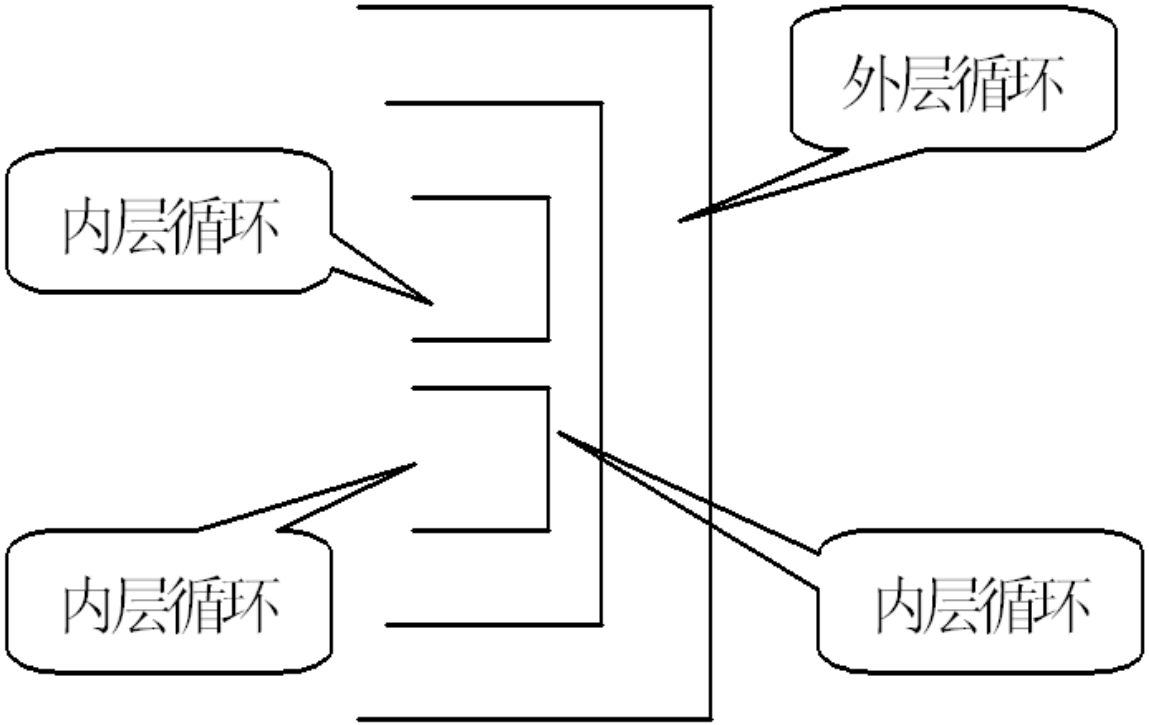
实例源码

```
1  #define CONST  1.42e8
2  int      n;
3  double term = 1, sum = 1;
4  for (n=2; n<=64; n++)
5  {
6       term *= 2;
7       sum += term;          /*作累加运算*/
8  }
9  printf("sum=%e\n", sum);          /*总麦粒数*/
10 printf("volum=%e\n", sum/CONST);  /*体积数*/
```

嵌套的循环体

- 在嵌套的各层循环体中，使用复合语句（即用一对大花括号将循环体语句括起来）保证逻辑上的正确性
- 内层和外层循环控制变量不应同名，以免造成混乱
- 嵌套的循环最好采用右缩进格式书写，以保证层次的清晰性
- 循环嵌套不能交叉，即在一个循环体内必须完整的包含着另一个循环

合法的嵌套循环



合法的嵌套循环

打印九九表

- 编程输出如下形式的乘法九九表

1	2	3	4	5	6	7	8	9
–	–	–	–	–	–	–	–	–
1	2	3	4	5	6	7	8	9
2	4	6	8	10	12	14	16	18
3	6	9	12	15	18	21	24	27
4	8	12	16	20	24	28	32	36
5	10	15	20	25	30	35	40	45
6	12	18	24	30	36	42	48	54
7	14	21	28	35	42	49	56	63
8	16	24	32	40	48	56	64	72
9	18	27	36	45	54	63	72	81

九九表源码

```
1       int  m, n;
2       for (m=1; m<10; m++)
3           printf("%4d", m);          /*打印表头*/
4           printf("\n");
5       for (m=1; m<10; m++)
6           printf("    -");
7           printf("\n");
8       for (m=1; m<10; m++) {
9           for (n=1; n<10; n++){
10              printf("%4d", m*n);
11          }
12          printf("\n");
13      }
```

打印九九表下三角

- 将上例输出格式改成如下的下三角格式打印

```
1   2   3   4   5   6   7   8   9
-   -   -   -   -   -   -   -   -
1
2   4
3   6   9
4   8   12  16
5   10  15  20  25
6   12  18  24  30  36
7   14  21  28  35  42  49
8   16  24  32  40  48  56  64
9   18  27  36  45  54  63  72  81
```

马克思手稿中的趣味学题–暴力计算

- 有30个人，其中有男人、女人和小孩，在一家饭馆里吃饭共花了50先令，每个男人各花3先令，每个女人各花2先令，每个小孩各花1先令，问男人、女人和小孩各有几人？

$$\begin{cases} x + y + z = 30 \\ 3x + 2y + z = 50 \end{cases}$$

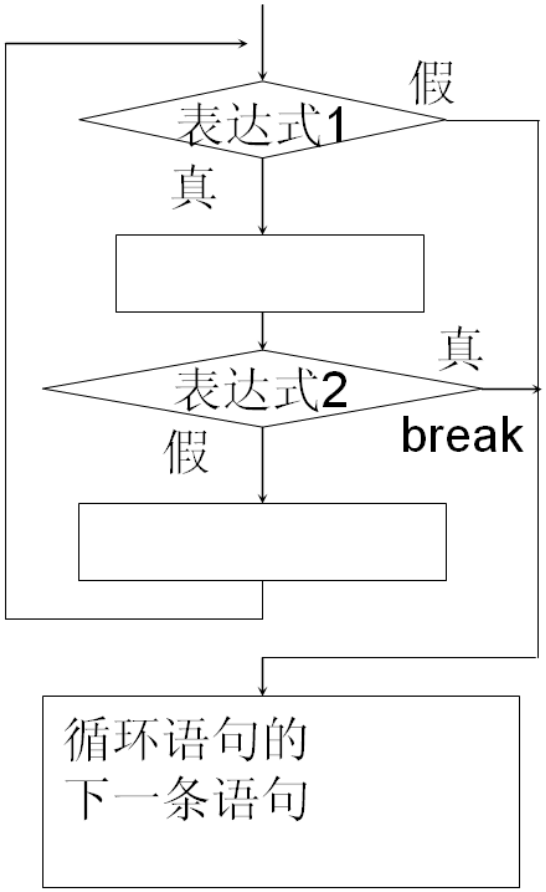
流程的转移控制

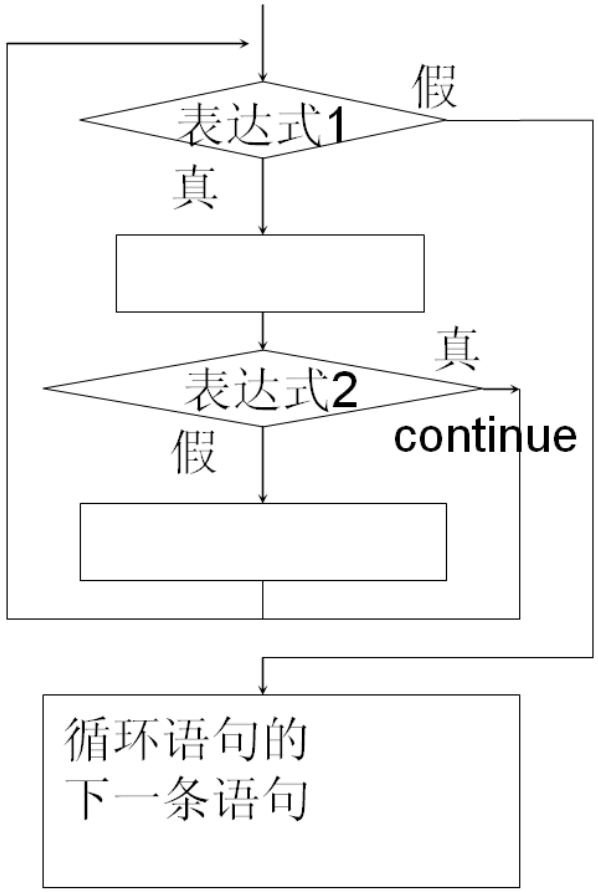
- 循环级别控制
  - `break`语句
  - `continue`语句
- 函数级别控制
  - `return`语句
- 程序级别控制
  - 标准库函数void exit(int status)

break和continue

- 在for、while、do-while循环进行内部放置
- break，退出循环
- continue，中断此次循环的执行，开始下一次循环
- break和continue少用为妙
  - 它们增加了循环执行的分支，break更增加了循环的出口
  - 它们可以用来处理程序异常，而尽量不要用来处理正常流程

break和continu区别





exit函数

- void exit(int status)
- 通常，status为零代表程序正确完成。
- 作用是终止整个程序的执行，强制返回操作系统
- 调用该函数需要包含头文件<stdlib.h>

去掉字符串中的空格

```
1   int ch;
2
3   while ((ch = getchar()) != '!')
4   {
5       if (ch == ' ')
6       {
7           continue;
8       }
9       putchar(ch);
10  }
11  printf("\nAll spaces have been removed\n");
```

goto与标号

- 标号举例
  - Error:
  - 同变量、函数的命名规则一样，后面加上一个冒号，一般顶格书写
- goto举例
  - goto Error;

使用goto的原则

- 使用之后，程序仍然是单入口，单出口
- 不要使用一个以上的标号
- 不要用goto往回走，要向下走
- 不要让goto制造出永远不会被执行的代码

糟糕的goto

```
1  START_LOOP:
2  if (fStatusOk) {
3      if (fDataAvaiable) {
4          i = 10;
5          goto MID_LOOP;
6      } else {
7          goto END_LOOP;
8      }
9  } else {
10     for (i = 0; i < 100; i++) {
11 MID_LOOP:
12     }
13     goto START_LOOP;
14 }
```

判断是否是质数-方法1

```
1  int  m, i, k;
2  printf("Please enter a number:");
3  scanf("%d", &m);
4  k = sqrt(m);
5  for (i=2; i<=k; i++){
6      if (m % i == 0){
7          printf("No!\n");
8          goto end;
9      }
10 }
11 printf("Yes!\n");
12 end:
13 printf("Program is over!\n");
```

判断是否是质数-方法2

```
1  int  m, i, k;
2  printf("Please enter a number:");
3  scanf("%d", &m);
4  k = sqrt(m);
5  for (i=2; i<=k; i++){
6      if (m % i == 0)
7          break;
8  }
9  if (i > k)
10     printf("Yes!\n");
11 else
12     printf("No!\n");
13 printf("Program is over!\n");
```

判断是否是质数-方法3

```
1  int  m, i, k, flag = 1; /*标志变量flag初值为1*/
2  printf("Please enter a number:");
3  scanf("%d",&m);
4  k = sqrt(m);
5  for (i=2; i<=k && flag; i++){
6      if (m % i == 0)
7          flag = 0;
8  }
9  if (flag)
10 printf("Yes!\n");
11 else
12 printf("No!\n");
13 printf("Program is over!\n");
```

错误分类

- 编译错误
  - 警告
  - 错误
- 运行错误
  - 本程序执行非法操作!
- 逻辑错误
  - 程序测试
  - 补丁

```
1  int i
2  /* compile error */
3
4  int i = 0;
5  5/i;
   /* runtime error */
6
7  int i = 0;
8  if(i=0)
9  {
10    printf("i is 0");
11    /* logical error */
12 }
```

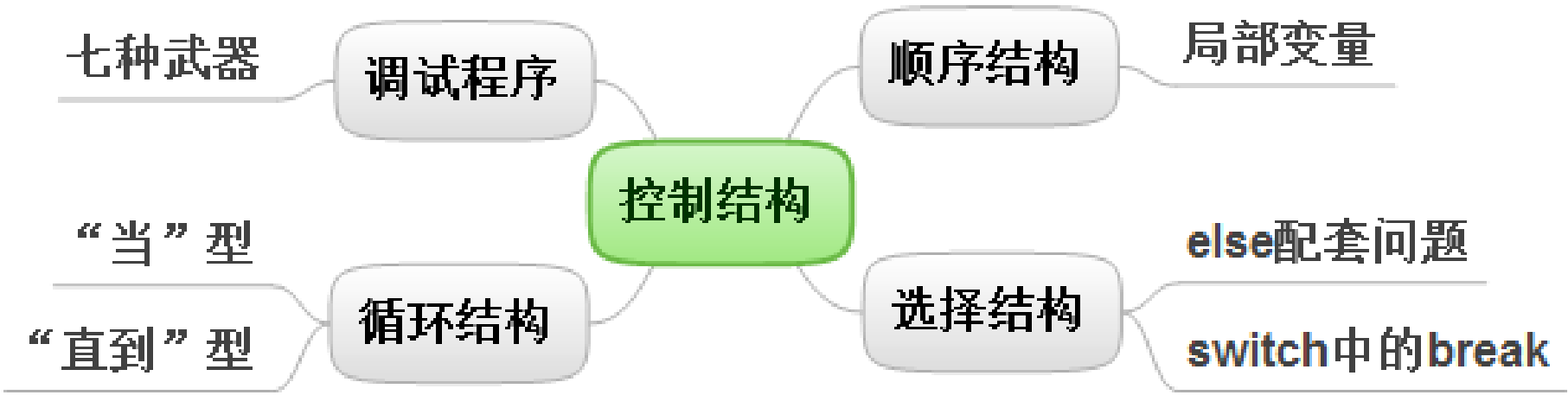
调试的七种武器

- 蒙汗药–断点
- 时间机器–单步执行
- 手术刀–监视窗
- 显微镜–内存影响
- 病例–函数调用栈
- 防火墙–assert
- 板砖–fprintf

调试的实例

```
1  int i ;
2  int a[10];
3  for(i = 0;i<=10;i++)
4  {
5      a[i] = 0;
6  }
```

总结


控制结构
调试程序
七种武器
顺序结构
局部变量
循环结构
“当”型
“直到”型
选择结构
else配套问题
switch中的break

谢谢大家，欢迎提问！