

计算机程序设计基础（1）

06 程序设计结构（下）

清华大学电子工程系

杨昉

E-mail: fangyang@tsinghua.edu.cn



● 语句和程序结构

- 语句：语句介绍；**结构化**和非结构化程序
- 基本程序结构：**顺序**结构、**分支**结构、**循环**结构可以构成所有程序

● 顺序结构设计

- **顺序**结构介绍、符合语句、顺序结构举例

● 分支结构设计

- **if语句**：简单if语句和if-else语句、if-else语句嵌套
- **条件运算符**：运算优先级、结合顺序
- **switch结构**：常量表达式、break作用

课程回顾:依照结构和规则以语句为单位进行结构化程序设计



类型	定义	说明
顺序结构	语句1; 语句2; 语句3;	顺序执行相关语句
分支结构 (if语句)	<pre>if(条件1) 语句1; else if(条件n) 语句n; else 语句n+1;</pre>	if语句: if (条件) 语句; if+复合语句,约束范围 if-else语句: if (条件) 语句1; else 语句2; if-else可以 嵌套 , 条件设置 和 顺序 需注意
分支结构 (条件运算符)	表达式1 ? 表达式2: 表达式3	方便、紧凑的分支语句,结合方向 从右到左 优先级 比赋值运算符高 ,比 关系、算术运算符低
分支结构 (switch结构)	<pre>switch(表达式) { case 常量表达式1: 语句1; ... case 常量表达式n: 语句n; default: 语句n+1; }</pre>	case 判断条件 , 语句n 执行命令 , default 备选 常量表达式必须是 整形 或 字符型 ,且 互不相同 , 但 顺序任意 ; default 可以没有 ; 内部顺序执行, 但break可从语句中 跳转出来 ,且需结合使用
复合语句: {...}实现, }后 不加分号 , 变量 生命周期 , 变量的 掩蔽		



6.1 循环结构

- 介绍
- while语句 与do-while语句
- 对键盘输入的讨论
- for语句
- 循环的嵌套

6.2 程序设计举例

- 列举与试探
- 密码问题
- 方程求根问题
- 级数求和问题

6.1 循环结构



6.1 循环结构



- 介绍：当型循环与直到型循环
- while语句与do-while语句
 - ✓ while语句
 - ✓ do-while语句
- 对键盘输入的讨论
- for语句
- 循环的嵌套
 - ✓ break语句
 - ✓ continue语句

循环结构：介绍



- 为何需要循环结构

- 顺序结构和分支结构不能解决某些问题：结构多次重复的

计算10!	$= 10 \times 9 \times 8 \times 7 \times 6 \times 5 \times 4 \times 3 \times 2 \times 1$	
计算100!	$= 100 \times 99 \times 98 \times \dots \times 3 \times 2 \times 1$	

- 重复性操作需要更方便的实现方式：循环结构应运而生

- 计算机非常适合做机械性的重复操作，可以高效地运行循环结构

循环结构：介绍

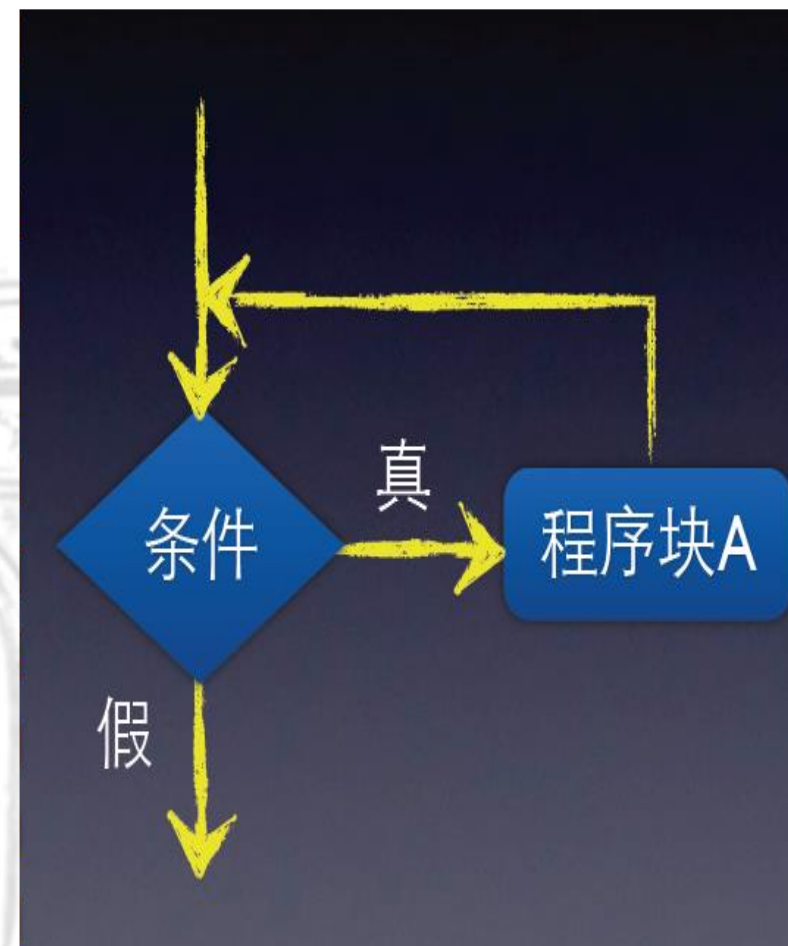


- **循环结构**：反复执行某一程序块的过程

- 进入循环结构，判断循环条件
- 如果循环条件的结果为**真(非0)**，则**执行程序块A**的操作，即循环一次，然后再次判断循环条件
- 当循环条件为**假(0)**时，**循环结束**

- 分为两种类型

- **当型**循环和**直到型**循环



循环结构：介绍



● 当型循环

- 先判断条件，当条件**满足**(即逻辑表达式的值为真)时，**执行循环体**中所包括的操作
- 当循环体执行完后，将再次判断条件，直到条件**不满足**(即逻辑表达式的值为假)为止，从而**退出循环结构**
- 如果在**开始**执行这个循环结构时条件就不满足，则当型循环结构中的循环体**一次也不执行**

当条件满足

循环体



● 直到型循环

- 首先执行循环体，然后判断条件（即计算逻辑表达式），如果条件满足（即逻辑表达式值为真），则退出循环结构
- 如果条件不满足（即逻辑表达式值为假），则继续执行循环体
- 由于首先执行循环体，然后再判断条件，因此，其循环体至少要执行一次。这是直到型循环结构与当型循环结构最明显的区别

循环体

直到条件满足



● 循环结构组成要素

- 循环条件设计：循环次数有限（计数控制）和循环次数未知（事件控制）
- 循环体设计：循环体如果包含一条以上语句时，则应该用花括号括起，构成复合语句
- 在循环体语句中，一定要有改变循环条件的语句，使循环能够终止。否则，将成为死循环

循环结构：介绍



● C语言提供的循环结构语句

□ while语句：

```
1 while (i<100) {  
2     sum += i;  
3     i++;  
4 }
```

□ do-while语句：

```
1 do {  
2     k++;  
3     s = s*k;  
4 } while (k<n);
```

□ for语句：

```
1 for (int i=1; i<=100; i++) {  
2     sum += 6;  
3 }
```



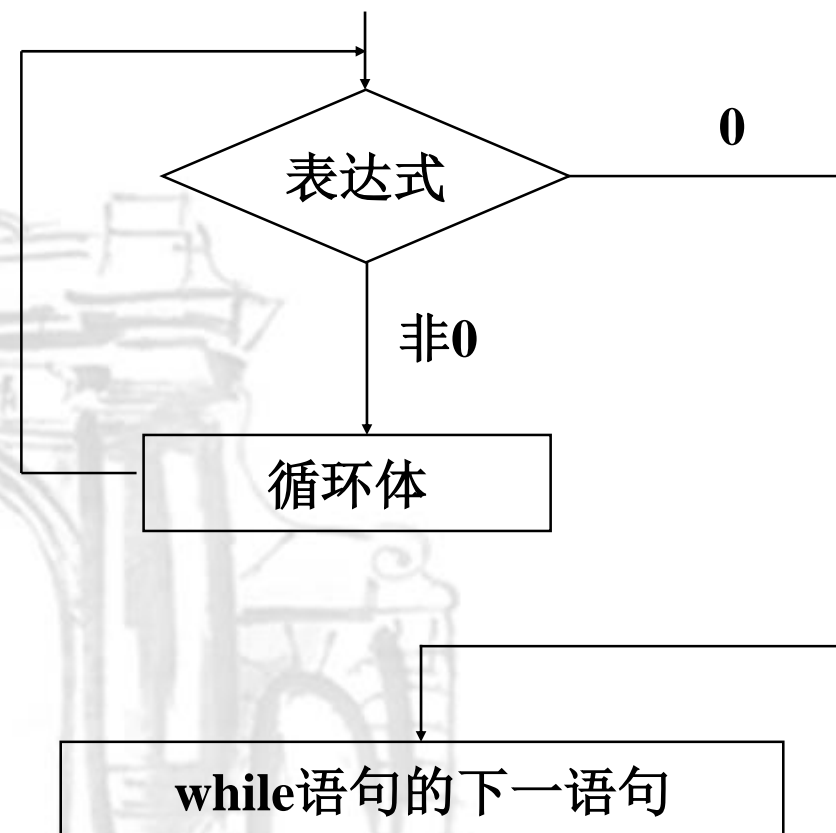
分别代表什
么意思呢？

循环结构：while语句



● while语句执行结构

- 首先判断条件表达式
- 条件为真(非0)进入循环体，执行内部的命令
- 条件为假(0)则跳出循环体，执行循环后面的语句
- 在用while语句构成循环结构的时候，在循环体内一定要有改变"表达式"（循环条件）值的语句，否则将造成死循环（即表达式值恒为1）



循环结构：while语句



- while语句表达式

while (表达式) 循环体语句;

- 例6-1：while语句功能

- 实现当型循环程序
- 首先计算表达式 $i < 100$ 的值，当 $i < 100$ 时，执行循环体语句sum和i的更新
- 执行完循环体中所有的语句后，再次计算表达式 $i < 100$ 的值
- 只有当 $i \geq 100$ 时才退出循环体，执行循环结构后面的printf语句

```
1 #include <stdio.h>
2 void main() {
3     int i;
4     int sum;
5     sum = 0; i=1;
6     while (i<100){
7         sum += i;
8         i++;
9     }
10    printf("sum = %d\n", sum);
11 }
```


循环结构：while语句



● 例6-2：循环变量需要设置边界值

□ 计算并输出下列级数和： $\text{sum} = 1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \dots + \frac{1}{50}$

```
1 #include <stdio.h>           //包含头文件"stdio.h"
2 void main() {                 //定义main函数，这是程序的主体
3     int n;
4     double sum;               //定义double型数sum
5     sum = 1.0; n=1;           //sum设为1, n设为1
6     while (n<50) {            //边界值
7         n++;
8         sum += 1.0/n;         //用1.0除为了确保结果是浮点数
9     }
10    printf("sum = %lf\n", sum); //打印sum的值
11 }
```

sum = 4.499205
请按任意键继续.....

循环结构：while语句



● 例6-2中注意浮点数除法

□ 以下几种计算方式将得到浮点数结果

1	sum += 1.0/n;
2	sum += 1/(double)n;
3	sum += (double)1/n;

□ 以下几种计算方式将得到整型数结果（向下取整）：因为1和n都是整型变量，当 $n > 1$ 时， $1/n$ 的值总为0

1	sum += 1/n;
2	sum += (double)(1/n);

循环结构：while语句



● 例6-3：从键盘输入若干学生成绩

□ 对成绩不及格（60分以下）的学生人数进行计数，直到输入的成绩为负为止，最后输出成绩不及格的学生人数

```
1  #include <stdio.h>           //包含头文件"stdio.h"
2  void main() {                 //定义main函数，这是程序的主体
3      int count;
4      float grade;              //定义float型数grade
5      count = 0;                //count设为1
6      scanf("%f", &grade);
7      while (grade>=0.0) {       //边界值
8          if(grade<60.0) count++;
9          scanf("%f", &grade);
10     }
11     printf("count = %d\n", count); //打印count的值
12 }
```

```
76 87 54 32 99 -1
count = 2
请按任意键继续.....
```

循环结构：while语句



● 例6-4：求 π 的近似值： $4 \times (1 - \frac{1}{3} + \frac{1}{5} + \dots + \frac{(-1)^k}{2k+1})$

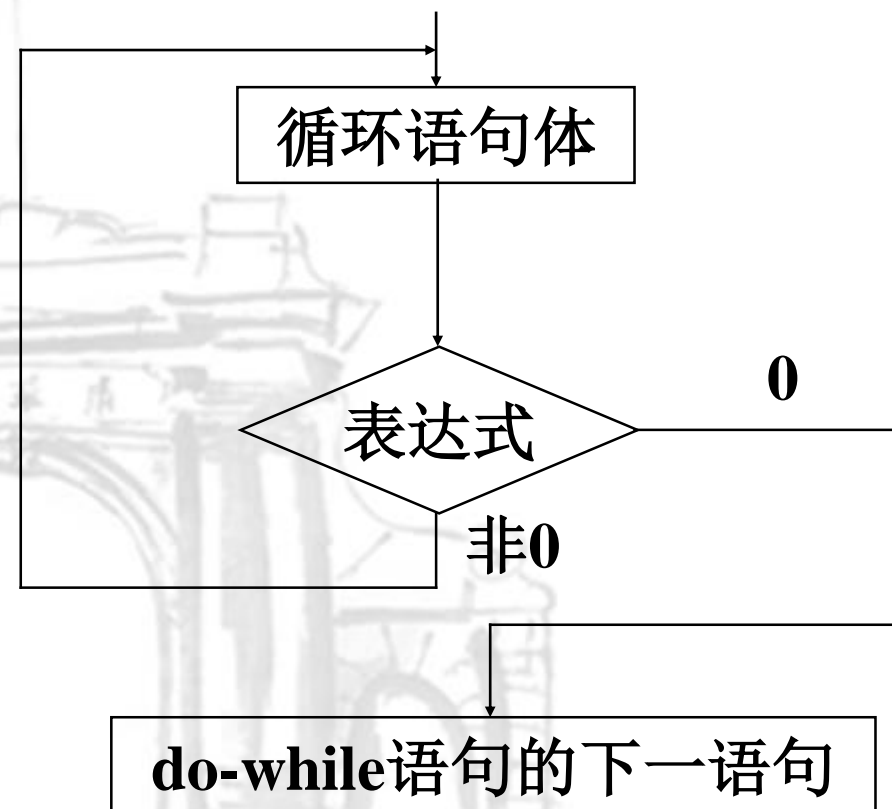
```
1 #include <stdio.h>           //包含头文件"stdio.h"
2 #include <math.h>            //包含头文件"math.h"
3 void main() {                //定义main函数，这是程序的主体
4     int s = 1, n = 1;
5     double t=1, pi = 0;      //定义double型数
6     while (fabs(t)>1e-7){     //边界值
7         pi += t;
8         n += 2;
9         s = -s;
10        t = (double)s/n;
11    }
12    pi *= 4;                  //得到近似的pi
13    printf( "pi = %lf\n" , pi); //打印pi的近似值
14 }
```

循环结构：do-while语句



● do-while语句执行结构

- 首先执行内部命令
- 执行完命令后进行条件判断
- 条件为真(非0)再次进入循环体，并执行内部的命令
- 条件为假(0)则跳出循环体，执行循环后面的语句



循环结构：do-while语句



- while语句表达式

do 循环体语句 while (表达式);

- 例6-5：do-while语句功能

- 实现直到型循环程序
- 首先执行循环体语句，更新sum和i的值，然后判断表达式值 $i < 101$
- 若 $i < 101$ 成立，则再次执行循环体
- 如此循环，直到 $i \geq 101$ 为止

```
1  #include <stdio.h>
2  void main() {
3      int i=0;
4      int sum;
5      sum = 0;
6      do {
7          sum += i;
8          i++;
9      } while(i<101);
10     printf("sum = %d\n", sum);
11 }
```

循环结构：do-while语句



● do-while和while语句的**辨析**

- do-while语句 在判断条件是否成立之前，先执行循环体语句一次
- while语句 则是先判断条件是否成立，如果条件成立才执行循环体
- while语句的循环体可能一次都不执行，而do-while语句的循环体至少被执行一次，这是while语句和do-while语句的**根本区别**
- 在有些问题中，如果其重复的操作(即循环体)有可能一次也不执行（即开始时条件就不满足），则要用while语句来处理，一般不用do-while语句来处理

循环结构：do-while语句



- 例6-6：计算并输出下列级数和，直到某项的绝对值小于 10^{-4} 为止

$$\text{sum} = 1 - \frac{1}{3} + \frac{1}{5} + \dots + \frac{(-1)^k}{2k+1}$$

```
1 #include <stdio.h>           //包含头文件"stdio.h"
2 void main() {                 //定义main函数，这是程序的主体
3     int k, f;
4     double sum, d;            //定义double型数sum
5     sum = 1.0; k=0; f=1;      //sum设为1.0, k设为0, f设为1
6     do {                      //边界值
7         k++; f = -f;
8         d = 1.0/(2*k+1);      //用1.0除为了确保结果是浮点数
9         sum += f*d;
10    } while(d>=1.0e-4);
11    printf("sum = %lf\n", sum); //打印sum的值
12 }
```

sum = 0.785448
请按任意键继续.....

循环结构：while语句和do-while语句



- 例6-7：下列C程序的功能是计算并输出n!（阶乘）值，其中n从键盘输入

```
1 #include <stdio.h>
2 void main() {
3     int n, k;
4     double s;
5     printf("input n:");
6     scanf("%d", &n);
7     k=1; s=1.0;
8     while(k<n) {
9         k++;
10        s = s*k;
11    }
12    printf("n! = %lf\n", s);
13 }
```

当型

```
1 #include <stdio.h>
2 void main() {
3     int n, k;
4     double s;
5     printf("input n:");
6     scanf("%d", &n);
7     k=0; s=1.0;
8     do {
9         k++;
10        s = s*k;
11    } while(k<n);
12    printf("n! = %lf\n", s);
13 }
```

直到型

循环结构：while语句和do-while语句



- 不管是用while语句还是用do-while语句实现循环结构
 - 在循环体内部必须要有能改变条件(即逻辑表达式值)的语句，否则将造成死循环
 - 例如例6-2将循环体内的条件改变为以下形式，则程序变成死循环，无法跳转

```
1  #include <stdio.h>           //包含头文件"stdio.h"
2  void main() {                //定义main函数，这是程序的主体
3      int n=1;
4      double sum=1.0;          //定义double型数sum
5      while (n<50) {           //边界值
6          n++;                //迭代变量n保持不变
7          sum += 1.0/n;         //用1.0除为了确保结果是浮点数
8      }
9      printf("sum = %lf\n", sum); }
```


循环结构：死循环



● 死循环

□ 你能让一个程序员一天到晚呆在淋浴房里吗？

给他一瓶洗发香波，上面写着：

```
while(1) {  
    涂抹香波;  
    温水冲洗;  
}
```



循环结构：对键盘输入的讨论



● 例6-8：编写一个C程序实现如下功能

- 从键盘输入一个英文字母，如果输入的英文字母为"y"或"Y"，则输出"yes!"
- 如果输入的英文字母为"n"或"N"，则输出"no!"

```
1  #include <stdio.h>
2  void main() {
3      char ch;
4      printf("Please input (y/n)?");
5      scanf("%c", &ch);
6      if (ch=='y' || ch=='Y')
7          printf("Yes!\n");
8      else
9          printf("No!\n");
10 }
```

循环结构：对键盘输入的讨论



Please input (y/n)? y
Yes!
请按任意键继续……

Please input (y/n)? Y
Yes!
请按任意键继续……

Please input (y/n)? n
No!
请按任意键继续……

Please input (y/n)? a
No!
请按任意键继续……

● 例6-8的输出结果显然不对

- ❑ 原题中要求只有读入单个的Y或者y输出Yes；只有读入单个的N或者n才输出No
- ❑ 在上一页的实现中，输入a也会输出No，不合题意
- ❑ 而当输入为一个字符串，例如yasfd时，下一次程序的执行会受到缓冲区中字符的影响

循环结构：对键盘输入的讨论



- 在编写程序时经常需要对键盘的字符输入进行读取
- 但是因为缓冲区的存在，有时用`getchar`得到的字符并不一定是此刻我们输入的字符，这就需要采用特殊的方法清空缓冲区再输入
 - 采用循环语句逐个字符地清空缓冲区

```
while(getchar() != '\n')
```
 - 关于清空输入缓冲区还有更方便简单的方法，在后续章节中会讲到用`fflush(stdin)`清空标准输入缓冲区

循环结构：对键盘输入的讨论



- 由缓冲区字符残留导致后续收集的字符出现错误，正确的修改方式应该为

```
1 #include <stdio.h> //包含头文件"stdio.h"
2 void main() {
3     char ch, ch2;
4     do { //采集字符直到ch为y/n/N/Y，或者ch2为换行符
5         printf("Please input (y/n)?");
6         scanf("%c", &ch);
7         scanf("%c", &ch2);
8         if (ch2!='\n') //如果ch2不是换行符，则循环清空缓冲区字符
9             while(getchar() != '\n');
10    } while(ch2!='\n' || (ch!='y' && ch!='n' && ch!='Y' && ch!='N'));
11    if (ch == 'y' || ch == 'Y') printf("Yes!\n");
12    else printf("No!\n");
13 }
```


循环结构：逻辑运算定律



● 基本运算：

□ 逻辑与(\wedge)、逻辑或(\vee)、逻辑非(\neg)、逻辑异或(\oplus)

□ C语言提供逻辑关系符： $\&\&$ 、 $||$ 、 $!$

● 基本定律

□ 交换律： $A \wedge B = B \wedge A$ ， $A \vee B = B \vee A$ ， $A \oplus B = B \oplus A$

□ 结合律： $A \wedge (B \wedge C) = A \wedge B \wedge C$ ， $A \vee (B \vee C) = A \vee B \vee C$

□ 分配律： $(A \vee B) \wedge C = (A \wedge C) \vee (B \wedge C)$ ， $A \vee (B \wedge C) = (A \vee B) \wedge (A \vee C)$

□ 吸收率： $A \vee (A \wedge B) = A$ ， $A \wedge (A \vee B) = A$ ， $A \vee (\neg A \wedge B) = A \vee B \dots$

□ 反演律： $\neg(A \wedge B) = \neg A \vee \neg B$ ， $\neg(A \vee B) = \neg A \wedge \neg B$

循环结构：对键盘输入的讨论



● 键盘输入问题：

□ 第二个字符ch2是'\n' 且 第一个字符为'n', 'N', 'y', 'Y'

□ 正问题： $ch2 == '\backslash n' \ \&\& \ (ch == 'y' \ || \ ch == 'n' \ || \ ch == 'Y' \ || \ ch == 'N')$

□ 反问题： $ch2 != '\backslash n' \ || \ \neg (ch == 'y' \ || \ ch == 'n' \ || \ ch == 'Y' \ || \ ch == 'N')$

$ch2 != '\backslash n' \ || \ (ch != 'y' \ \&\& \ ch != 'n' \ \&\& \ ch != 'Y' \ \&\& \ ch != 'N')$

□ 逻辑表达式：

$(ch2 == '\backslash n' \ \&\& \ (ch == 'y' \ || \ ch == 'n' \ || \ ch == 'Y' \ || \ ch == 'N')) == 0$

循环结构：事件控制循环



- 例6-9：从键盘连续输入字符，直到输入"回车"符为止，统计输入的字符个数

□ 未知循环次数的循环条件（事件控制循环）

□ getchar()/getch()函数使用和区别

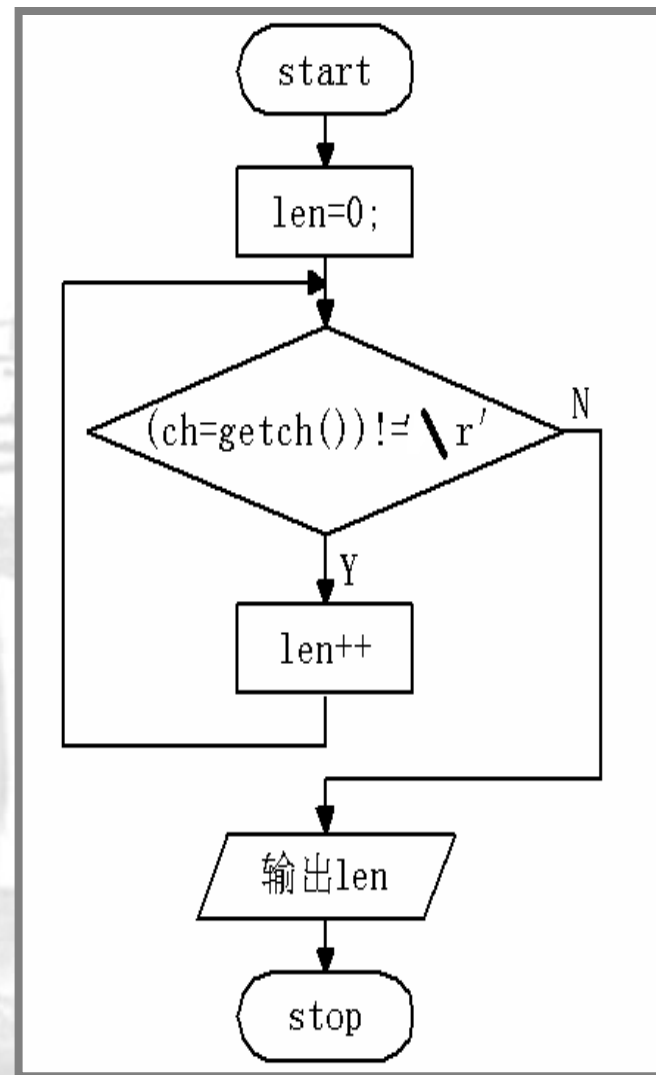
- `char c; c=getchar();` 或 `c=getch();`
- `c=getch();` 头文件是 conio.h，而不是 stdio.h
- Windows下ENTER键会产生两个转义字符`\r\n`，`getchar()` 返回10 (即 \n)，`getch` 返回13 (即 \r)

循环结构：事件控制循环



● 例6-9：采用事件控制循环，结构图如右

```
1 #include <stdio.h>
2 #include <conio.h>
3 void main() {
4     char ch;
5     int len=0;
6     printf("Type in a sentence, then press <Enter>\n");
7     while ((ch=getchar())!='\n') {
8         len++;
9     }
10    printf("\n Sentence is %d characters long.\n", len);
11 }
```



循环结构：计数控制循环

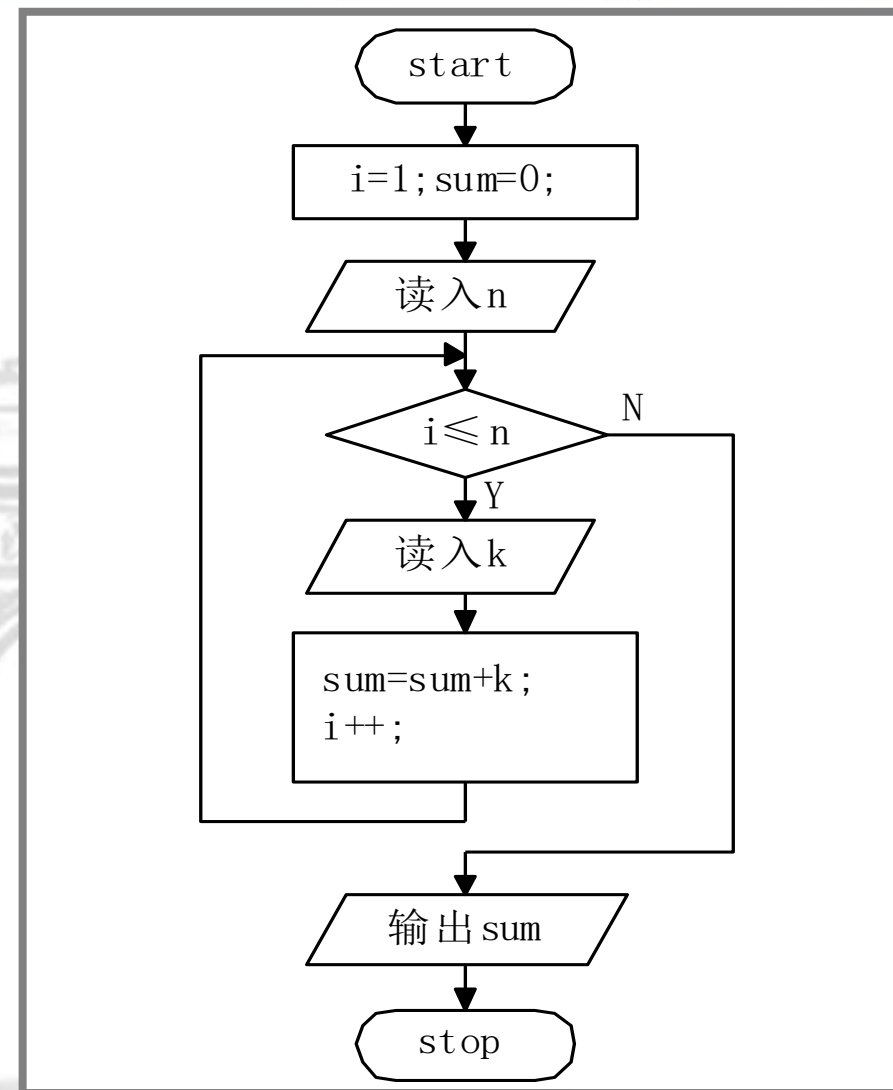


● 例6-10：从键盘连续输入 n 个数，求其和

□ 循环条件（**计数控制循环**）

□ 循环体构造：**计数器** (i) 和 **累加器** (sum) 的概念

□ 流程图如右图所示：确定数字个数 n ，再依次读入数据并累加，最后输出求和结果



循环结构：计数控制循环



● 例6-10用当型循环结构实现

```
1  #include <stdio.h>    //包含头文件
2  void main() {
3      long int sum=0;
4      int i=1, n, k;
5      printf("Input n:");
6      scanf("%d", &n);
7      while (i<=n) {
8          scanf("%d", &k); //依次读入数据
9          sum += k;
10         i++;
11     }
12     printf("\n Sum is %d\n", sum);
13 }
```

```
Input n: 1
5
Sum is 5
请按任意键继续.....
```


循环结构：事件控制vs计数控制



● 练习6-1：写出下面程序的输出

```
1 #include <stdio.h>
2 void main() {
3     int a =21, b=133;
4     int temp;
5     while(a != 0) {
6         temp = b%a;
7         b = a;
8         a = temp;
9     }
10    printf("%d\n", b);
11 }
```

7
请按任意键继续. . .

这是事件控制循环还是计数控制循环？

循环结构：for语句



- for语句表达式

for (<表达式1> ; <表达式2> ; <表达式3>)

{循环体语句;}

- 表达式1： 循环变量赋初值

- 表达式2： 循环条件

- 表达式3： 循环变量增值

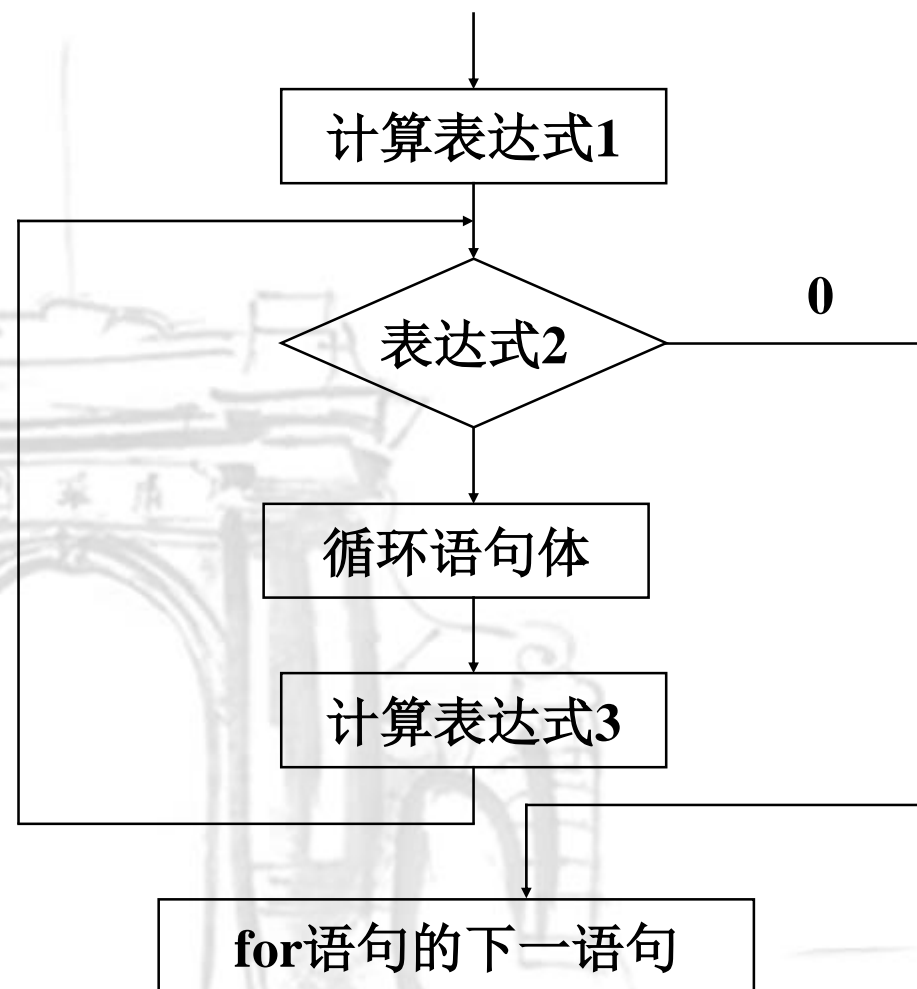
- 功能：更精简地实现 当型循环结构，不用在循环外单独定义 循环变量

循环结构：for语句



● for语句执行流程图

- 表达式1中内部对循环变量赋初始值
- 判断表达式2的条件，如果为是则进入循环体执行命令；否则跳出循环
- 在循环内部执行完命令后，根据表达式3更新循环变量，并进行下一次循环判断



循环结构：for语句



● 例6-11：用for循环语句编写计算100个6相加的程序

```
1 #include <stdio.h>
2 void main() {
3     int sum = 0;
4     int i;
5     for (i=1; i<=100; i++) {
6         sum += 6;
7     }
8     printf("\n Sum is %d\n", sum);
9 }
```

for语句实现

```
1 #include <stdio.h>
2 void main() {
3     int i=0;
4     int sum = 0;
5     while (i<100) {
6         sum += 6;
7         i++;
8     }
9     printf("\n Sum is %d\n", sum);
10 }
```

while语句实现

循环结构：for语句



- 在for语句中，语法上<表达式1>与<表达式3> 均可省略，但其中的两个";" 不能省略

1 等价结构

2 ① `for (int i=1; i<=100; i++) <循环体>`

3 ② `int i = 1;`

4 `for (; i<=100; i++) <循环体>`

5 ③ `int i = 1;`

6 `for (; i<=100;) <循环体; i++;>`

7 ④ `int i = 1;`

8 `while (i<=100) <循环体; i++;>`

- 在for语句中如果没有<表达式2>，则将构成死循环

循环结构：for语句



- 使用for语句一般解决事先知道循环的起始点和终止点及其循环次数的问题
- for循环本质上也是当型循环结构，只不过它对于事先可以确定环次数的问题特别方便

1	常用for结构
2	① <code>for (i=a; i<=b; i=i+步长) <循环体></code>
3	② <code>for (i=b; i>=a; i=i-步长) <循环体></code>

- 在for循环中，循环体可以是复合语句（即用一对花括号{ }括起来的语句组）

循环结构：for语句



● 例6-12：重写例6-2，采用for语句

□ 计算并输出下列级数和：
$$\text{sum} = 1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \cdots + \frac{1}{50}$$

```
1 #include <stdio.h>    //包含头文件"stdio.h"
2 void main() {          //定义main函数，这是程序的主体
3     double sum;        //定义double型数sum
4     sum = 1.0;          //sum设为0, n设为1
5     for (int n=2;n<=50;n++) {    //边界值
6         sum += 1.0/n;    //用1.0除为了确保结果是浮点数
7     }
8     printf("sum = %lf\n", sum); //打印sum的值
9 }
```

sum = 4.499205
请按任意键继续.....

课堂练习：for语句



● 练习6-2：重写例6-7，采用for循环计算n!

```
1 #include <stdio.h>
2 void main() {
3     int n, k;
4     double s;
5     printf("input n:");
6     scanf("%d", &n);
7     k=1; s=1.0;
8     while(k<n) {
9         k++;
10        s = s*k;
11    }
12    printf("n! = %lf\n", s);
13 }
```

当型

```
1 #include <stdio.h>
2 void main() {
3     int n, k;
4     double s;
5     printf("Input n:");
6     scanf("%d", &n);
7     s=1.0;
8     for (k=1; k<=n; k++) {
9         s = s*k;
10    }
11    printf("n! = %lf\n", s);
12 }
```

for循环

循环结构：循环的嵌套



- 在循环体中又包含另一个循环语句，称为循环嵌套
- 利用嵌套结构，可以组成更加复杂的程序，在多个维度上对变量进行遍历
- 在多重循环中，处于内部的循环称为内循环，处于外部的循环称为外循环
- 内循环必须完全嵌套于外循环中，内、外循环不能交叉；且内、外循环的循环控制变量不能重名

循环结构：循环的嵌套



- break语句的表达式： **break;**
- break功能
 - 跳出switch 结构
 - 退出当前循环结构,包括while、do-while和for循环结构
- break特点
 - 在循环结构中的break语句只是退出当前循环结构
 - 循环结构中的break语句是一个**非正常出口**，理论上是不符合结构化程序设计原则的
 - 建议在循环结构中尽量不用break语句退出

循环结构：循环的嵌套



● 例6-13：找出3位数中最大的5个素数

997 991 983 977 971

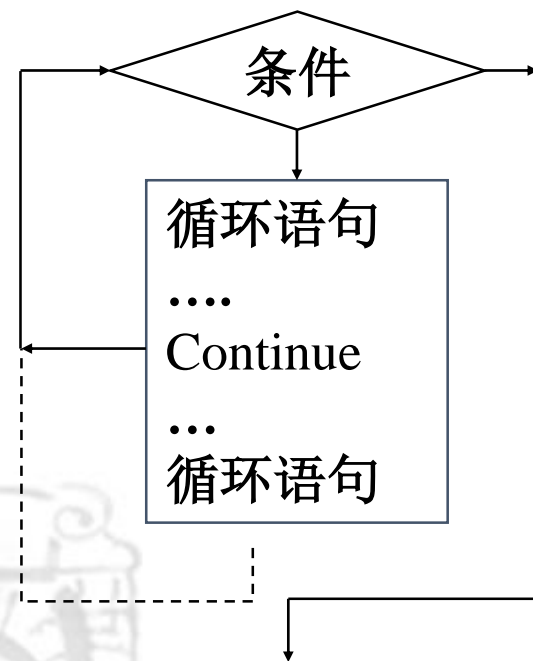
请按任意键继续……

```
1  #include <stdio.h>           //包含头文件"stdio.h"
2  #include <math.h>            //包含头文件"math.h"
3  void main() {
4      int j=0, n, k, i, flag;
5      for (n=999; n>=101; n=n-2) { //倒序遍历三位数
6          k = (int)sqrt((double)n); //最大约数
7          i = 2; flag = 0;
8          while ((i<=k)&&(flag==0)) { //判断素数
9              if (n%i==0) flag = 1;    //flag==0为素数
10             i++;
11         }
12         if (flag==0)
13             { j++; printf("%d", n); } //计算素数并输出
14         if (j==5) break;             // 输出5个后跳出
15     }
16     printf("\n");
17 }
```

循环结构：循环的嵌套



- continue语句的表达式：**continue;**
- 功能：结束本次循环执行，但不退出循环
- continue特点
 - 与循环语句连用，通常不单独使用
 - 利用continue语句可以在循环体的任何位置上结束本次循环而开始下一次的循环，破坏了循环结构的正常执行顺序
 - 严格来说，它是一个**不符合结构化原则**的语句



循环结构：循环的嵌套



● 例6-14：输出100 ~ 200之间所有能被7和9整除的自然数

```
1 #include <stdio.h>           //包含头文件
2 void main() {
3     int n;
4     for(n=100; n<=200; n++) { //遍历100~200的三位数
5         if ((n%7!=0) || (n%9!=0))
6             continue;         //找不到就进入下次循环
7         printf("%d\n", n);
8     }
9 }
```

```
if ((n%7==0)&&(n%9==0))
    printf("%d\n", n);
```

126
189
请按任意键继续.....

- 着眼我国发展阶段、环境、条件变化，要推动形成以**国内大循环**为主体、国内国际**双循环**相互促进的新发展格局。
——习近平



- 同样的过程不必重复进行
- 可以利用循环的方法，对程序进行抽象，获得更为精简的程序
- 循环在程序编写中必不可少



6.2 程序设计举例

- 列举与试探
- 密码问题
- 方程求根问题
- 级数求和问题

程序设计举例：列举与试探法



- 列举法又称穷举法，基本思想是：根据提出的问题，**列举所有可能的情况**，并用问题中给定的条件检验哪些是需要的，哪些是不需要的
- 列举法常用于解决"**是否存在**"或"**有多少种可能**"等类型的问题，例如求解不定方程的问题
- 例6-15：某参观团按以下限制条件从A， B， C， D， E五个地方中选定若干参观点：
 - 如果去A地，则必须去B地；

程序设计举例：列举与试探法



- D和E两地中只能去一地；
- B和C两地中只能去一地；
- C和D两地要么都去，要么都不去；
- 如果去E地，则必须去A和D地。问该参观团能去哪几个地方？
- 思路：用a、b、c、d、e五个整型变量，分别表示A、B、C、D、E是否去的状态。若变量值为1，则表示去该地；若变量值为0，则表示不去该地
- 程序设计：采用for循环遍历所有可能情况，并从中选择出符合题目要求的目标情况

程序设计举例：列举与试探法



程序复杂
情况太多
执行较慢

```
1 #include <stdio.h>
2 void main() {
3     int a, b, c, d, e;
4     for (a=0; a<=1; a++)
5         for (b=0; b<=1; b++)
6             for (c=0; c<=1; c++)
7                 for (d=0; d<=1; d++)
8                     for (e=0; e<=1; e++)
9                         if ((a&&b||!a) && d+e==1 && b+c==1 && (c+d==2||c+d==0) && (e&&a+d==2||!e)) {
10                             printf("A will %s go.\n", a? "":"not"); //条件表达式
11                             printf("B will %s go.\n", b? "":"not");
12                             printf("C will %s go.\n", c? "":"not");
13                             printf("D will %s go.\n", d? "":"not");
14                             printf("E will %s go.\n", e? "":"not");
15                         }
16 }
```


程序设计举例：列举与试探法



- 试探法在另外一些问题中，可能其列举量事先并不知道，只能从初始情况开始，往后逐步进行试探，直到满足给定的条件为止这就是逐步试探的方法，简称试探法
- 例6-16，某幼儿园按如下方法依次给A、B、C、D、E五个小孩发苹果。确定原来共有多少个苹果？每个小孩各得到多少个苹果？
 - 将全部苹果的一半再加二分之一一个苹果发给第一个小孩
 - 将剩下苹果的三分之一再加三分之一一个苹果发给第二个小孩
 - 将剩下苹果的四分之一再加四分之一一个苹果发给第三个小孩
 - 将剩下苹果的五分之一再加五分之一一个苹果发给第四个小孩
 - 将最后剩下的11个苹果发给第五个小孩

程序设计举例：列举与试探法



- 思路：设 x 是总苹果数，则

□ $a = (x + 1) / 2;$	// 第一个小孩分到的苹果数
□ $b = (x - a + 1) / 3;$	// 第二个小孩分到的苹果数
□ $c = (x - a - b + 1) / 4;$	// 第三个小孩分到的苹果数
□ $d = (x - a - b - c + 1) / 5;$	// 第四个小孩分到的苹果数
□ $e = x - a - b - c - d = 11;$	// 第五个小孩分到的苹果数

- 程序设计：采用for循环模拟发放过程，逐步试探可能的结果，直到获取满足条件的结果

程序设计举例：列举与试探法



```
1 #include <stdio.h>
2 void main() {
3     int n, flag, k, x, a, b, c, d, e; n = 11; flag = 1;
4     while(flag) { //进行试探
5         x = n; //保存当前试探值
6         flag = 0; //清除标志
7         for (k=1; k<=4&&flag==0; k++) //模拟四次发放过程
8             if ((n+1)%(k+1)==0) //小朋友得到整数个苹果
9                 n = n-(n+1)/(k+1); //计算余下苹果
10            else flag = 1; //不是整数则设置标志
11            if (flag==0 && n!=11) flag = 1; //试探不成功
12            n = x + 1; //下一次试探值
13    }
14    printf("Total number of apple=%d\n", x);
15    a = (x+1)/2; b = (x-a+1)/3;
16    c = (x-a-b+1)/4; d = (x-a-b-c+1)/5; e = 11;
17    printf("A=%d\nB=%d\nC=%d\nD=%d\nE=%d\n ", a, b, c, d, e);
18 }
```

```
Total number of apple=59
A=30
B=10
C=5
D=3
E=11
请按任意键继续.....
```

程序设计举例：密码问题



- 在报文通信中,为使报文保密,发报人往往要按一定规律将其加密,收报人再按约定的规律将其解密(即将其译回原文)
- 一种简单的加密方法
 - 将报文中的每一个英文字母转换为其后的第k个字母,而非英文字母不变。例如,当 $k=5$ 时,字母a转换为f,B转换为G等。由此可以看出,只需将该字母的ASCII码加上5 (k 的值) 即可
 - 在转换过程中,如果某大写字母其后的第 k 个字母已经超出大写字母Z,或某小写字母其后的第 k 个字母已经超出小写字母z,则将循环到字母表的开始。例如,大写字母V转换为A,大写字母Z转换为E,小写字母v转换为a,小写字母z转换为e等

程序设计举例：密码问题



● 例6-17：键盘输入一行字符，仅将其中英文字母加密输出

```
1 #include <stdio.h>
2 void main() {
3     char c; int k;
4     printf("input k:");
5     scanf("%d", &k);    //输入
6     getchar();          //吃掉缓存区回车
7     c = getchar();       //读取1个字符
8     while(c!='\n') {
9         if ((c>='a' && c<='z') || (c>='A' && c<='Z')) {
10             c += k;
11             if (c>'z' || (c>'Z' && c<='Z'+k)) c -= 26; //加密
12         }
13         printf("%c", c);
14         c = getchar();
15     } }
```

Input k:5
are you ready?
fwj dtz wjfid?
请按任意键继续.....

程序设计举例：密码问题



- 例6-18：解密过程是加密过程的逆过程，用下面程序实现

```
1  #include <stdio.h>
2  void main() {
3      char c; int k;
4      printf("input k:");
5      scanf("%d", &k);    //输入
6      getchar();          //吃掉缓存区回车
7      c = getchar();       //读取1个字符
8      while(c!='\n') {
9          if ((c>='a' && c<='z') || (c>='A' && c<='Z')) {
10             c -= k;
11             if ((c<'a' && c>='a' -k) || c<'A') c += 26; //解密
12         }
13         printf("%c", c);
14         c = getchar(); }
15 }
```

Input k:5
fwj dtz wjfid?
are you ready?
请按任意键继续.....

程序设计举例：方程求根问题



- **对分法**求方程实根：

- 假设非线性方程 $f(x)=0$ 的左端函数 $f(x)$ 在区间 $[a,b]$ 上连续
- 并满足： $f(a)f(b)<0$
- 则该非线性方程在区间 $[a,b]$ 上至少有一个实根

- 基本思想：

- 逐步缩小有根的区间，当这个区间长度减小到一定程度时，就取这个区间的中点作为根的近似值
- 取有根区间的中点，即令 $x=(a+b)/2$

程序设计举例：方程求根问题



- 若 $f(x)=0$ ，则 x 即为根，过程结束
- 若 $f(a)f(x)<0$ ，则说明实根在区间 $[a,x]$ 内，令 $b=x$
- 若 $f(b)f(x)<0$ ，则说明实根在区间 $[x,b]$ 内，令 $a=x$
- 若 $|a-b|<e$ （ e 为预先给定的精度要求），则过程结束， $(a+b)/2$ 即为根的近似值（已满足精度要求）
- 否则从第一步开始重复执行
- 如果在区间 $[a,b]$ 内有多个实根，则单独利用对分法只能得到其中的一个实根

程序设计举例：方程求根问题



- 在实际应用中，可以将逐步扫描与对分法结合起来使用，以便尽量搜索出给定区间内的所有实根
- 这种方法的重点如下：
 - 从区间左端点 $x=a$ 开始，以 h 为步长，逐步往后进行搜索
 - 对于在搜索过程中遇到的每一个子区间 $[x_k, x_{k+1}]$
 - 若 $f(x_k)=0$ ，则 x_k 为一个实根，且从 $x_k+h/2$ 开始往后再搜索
 - 若 $f(x_{k+1})=0$ ，则 x_{k+1} 为一个实根，且从 $x_{k+1}+h/2$ 开始往后再搜索

程序设计举例：方程求根问题



- 若 $f(x_k)f(x_{k+1}) > 0$ ，则说明在当前子区间内 **无实根或h选得过大**，放弃当前子区间，从 x_{k+1} 开始往后再搜索
 - 若 $f(x_k)f(x_{k+1}) < 0$ ，则说明在当前子区间 **内有实根**，此时 **利用对分法**，直到求得一个实根为止，然后从 x_{k+1} 开始往后再搜索
 - 在进行根的搜索过程中，要 合理选择步长，尽量避免根丢失
- 例6-19：用对分法求方程 $f(x) = x^2 - 6x - 1 = 0$ 在区间 $[-10, 10]$ 上的实根
- 即 $a = -10$ ， $b = 10$ 。取扫描步长 $h = 0.1$ ，精度要求 $\varepsilon = 1e-6$

程序设计举例：方程求根问题



```
1 #include <stdio.h>
2 #include <math.h>
3 #define F(x) ((x)*(x)-6.0*(x)-1.0)
4 void main() {
5     double a=-10.0, b= 10.0, h= 0.1, x1, y1, x2, y2, x, y;
6     int flag;
7     x1 = a; y1 = F(x1);
8     x2 = x1+h; y2 = F(x2);
9     while(x1<=b) {
10         if (y1*y2>0.0) //小区间两端点函数值同号，无根
11             { x1 = x2; y1 = y2; x2 = x1+h; y2 = F(x2); }
12         else { //小区间两端点函数值异号，有根
13             flag = 0;
14             while(flag==0) {
15                 x = (x1+x2)/2; //取小区中点
16                 if (fabs(x2-x1)<1e-6) { //满足精度要求
```

程序设计举例：方程求根问题



```
22     printf("x=%11.7f\n", x);
23     x1 = x+0.5*h; y1 = F(x1);           //搜索下一区间
24     x2 = x1+h; y2 = F(x2);
25     flag = 1;
26 }
27 else {                                 //不满足精度要求
28     y = F(x);
29     if (y1*y<0.0) {x2 = x; y2 = y;}
30     else {x1 = x; y1 = y;}
31 }
32 }
33 }
34 }
35 }
```

```
x=-0.1622776
x=6.1622780
请按任意键继续.....
```


程序设计举例：级数求和问题



- 例6-20：计算下列级数和，直到 $\frac{x^n}{n!} < 10^{-6}$ 。n和x从

键盘读入

$$S(x) = 1 + x + \frac{x^2}{2!} + \dots + \frac{x^n}{n!}$$

- 分析

- 设级数的第n项为： $T_n = \frac{x^n}{n!}$

- 则第n+1项： $T_{n+1} = \frac{x^{n+1}}{(n+1)!} = \frac{x^n}{n!} \times \frac{x}{n+1} = T_n \times \frac{x}{n+1}$

- 得到递推式： $T_{n+1} = T_n \times \frac{x}{n+1}$ ，且 $T_0 = 1$

程序设计举例：级数求和问题



● 由此得到下面的程序

```
1 #include <stdio.h>
2 void main() {
3     double s=1.0, t=1.0, x=2.0; //S=T_0, T_0=1
4     int n=1;
5     do {
6         t *= x/n; //T_n+1=T_n*x/(n+1)
7         s += t;
8         n++;
9     } while(t>1e-6);
10    printf("%12.6f %12.6f\n", s, exp(2.0));
11 }
```

7.389057 7.389056
请按任意键继续.....



● 循环结构

- 当型循环与直到型循环
- while语句与do-while语句
- 对键盘输入的讨论
- for语句
- 循环的嵌套：break和continue语句

● 程序设计举例

- 列举与试探
- 密码问题、方程求根问题、级数求和问题



● 第六讲作业

- 教材第七章，p.158习题2，7，9
- 教材第七章，p.159习题10
- 完成后将word文档或拍照提交到网络学堂



● 附加作业

- 输入一个整数 n ，输出数字 m ，其中 m 是斐波那契数列的第 n 项
- 输入一个正整数 n ，输出 n 的所有质因子
- 输出1-1000内的所有“完全数”，完全数指一个正整数，且等于其除自身外所有正因子之和，如6的因子为1，2，3，6， $6=1+2+3$ ；因此6是一个完全数



THANKS