

C语言程序设计

The C Programming Language



第4章 程序与流程控制

李春花

武汉光电国家研究中心

C语言程序的书写格式有规律吗？

C语言程序的基本结构有哪些？

设计C语言程序应从哪些方面思考问题？

编写一些简单的C语言程序有规律可循吗？

如何培养严谨的思维和良好的习惯？

例 求两个整数之和

```
#include <stdio.h>    //预处理
void main( )          //函数定义
{
    int a,b;           //变量说明
    int sum;
    scanf("%d%d",&a,&b); // 数据输入
    sum = a + b;        // 执行部分
    printf("sum=%d",sum); // 信息输出
}
```

这是一个典型的只包含主函数main() 的程序
main函数--程序入口函数

一般包含**七部分内容**:

- (1) 注释部分
- (2) 预处理块、**全局变量说明、函数声明等**
- (3) **函数定义部分**
- (4) 变量说明部分
- (5) 数据输入部分
- (6) 执行部分 (**核心**)
- (7) 信息输出部分

4.0

求两个整数之和的程序

若采用函数调用，则程序可修改为：

```
#include <stdio.h>           //预处理
int add(int x,int y);        //函数声明

void main( )                 //函数定义
{
    int a,b;                 //变量说明
    int sum;

    scanf("%d%d",&a,&b); //数据输入

    ● sum = add(a, b);       //执行部分

    printf("sum=%d",sum); //信息输出
}
```

```
//求和函数
//输入： 两个整数，
//返回： 和
int add(int x, int y) //函数定义
{
    int z;             // 变量说明

    z = x + y;         //执行部分

    return z;          //返回结果
}
```

包含主函数main() + 调用函数的程序



主要内容

复合语句

选择语句: if、switch

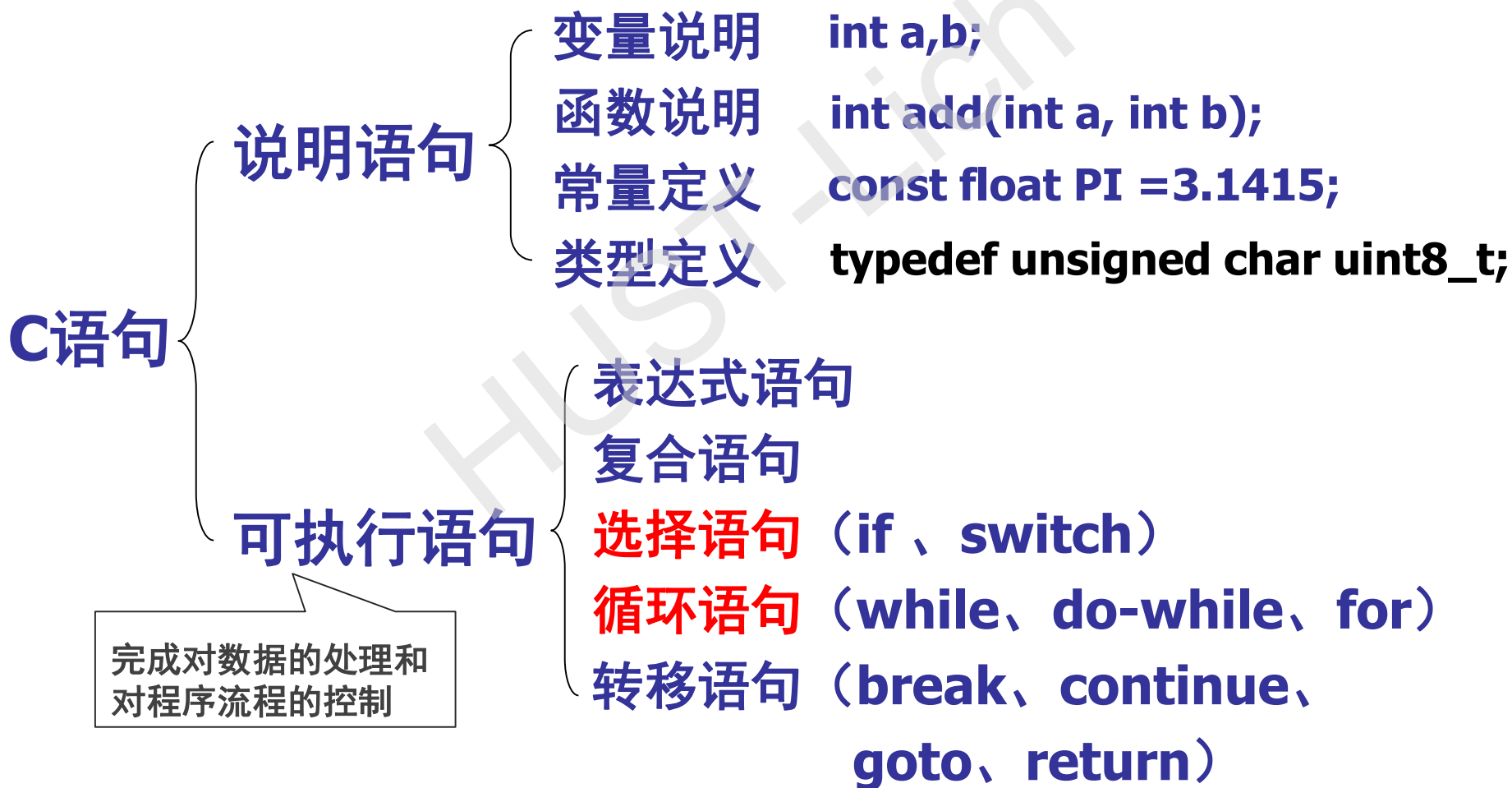
循环语句: while、for、do-while

转移语句: break、continue、goto

常用算法: 枚举、筛法、递推

4.1 语句分类

语句是程序的基本元素，程序中的各功能部分都是由一定含义的语句组成的





4.2 表达式语句

表达式语句：任何一个表达式加上一个分号就是一条语句

表达式； **//;** 一个语句结束的标志

只有分号而没有表达式的语句就叫空语句；空语句格式为：

; **//空语句**

例： **x = y + 1** **/* 赋值表达式 */**

x = y + 1; **/* 赋值表达式语句 */**

printf("hello"); **/* 函数调用语句 */**

; **/* 空语句 */**

4.3 复合语句

复合语句：将若干语句用括号{ }括起来就构成了复合语句

{

说明语句 //可无

执行语句

}

复合语句又称**程序块**

函数体也是一个程序块

复合语句在语法上相当于一个语句

函数体

```
#include<stdio.h>
```

```
int main( void )
```

```
{
```

```
int x=5,y=1,t;
```

```
if(x>y) {
```

```
    t=x;
```

```
    x=y;
```

```
    y=t;
```

```
}
```

```
printf ("x=%d,y=%d\n", x,y);
```

```
return 0;
```

```
}
```

复合语句

注意变量的作用域

变量的作用域

在复合语句内说明的变量，其作用域局限于该复合语句

```
int main(void)
{
    int x=1,y=2;
    {
        int x=2;
        printf ("x=%d, y=%d\n", x,y);    // x=2, y=1
    }
    printf ("x=%d, y=%d\n", x,y);    // x=1, y=1
    return 0;
}
```

函数体

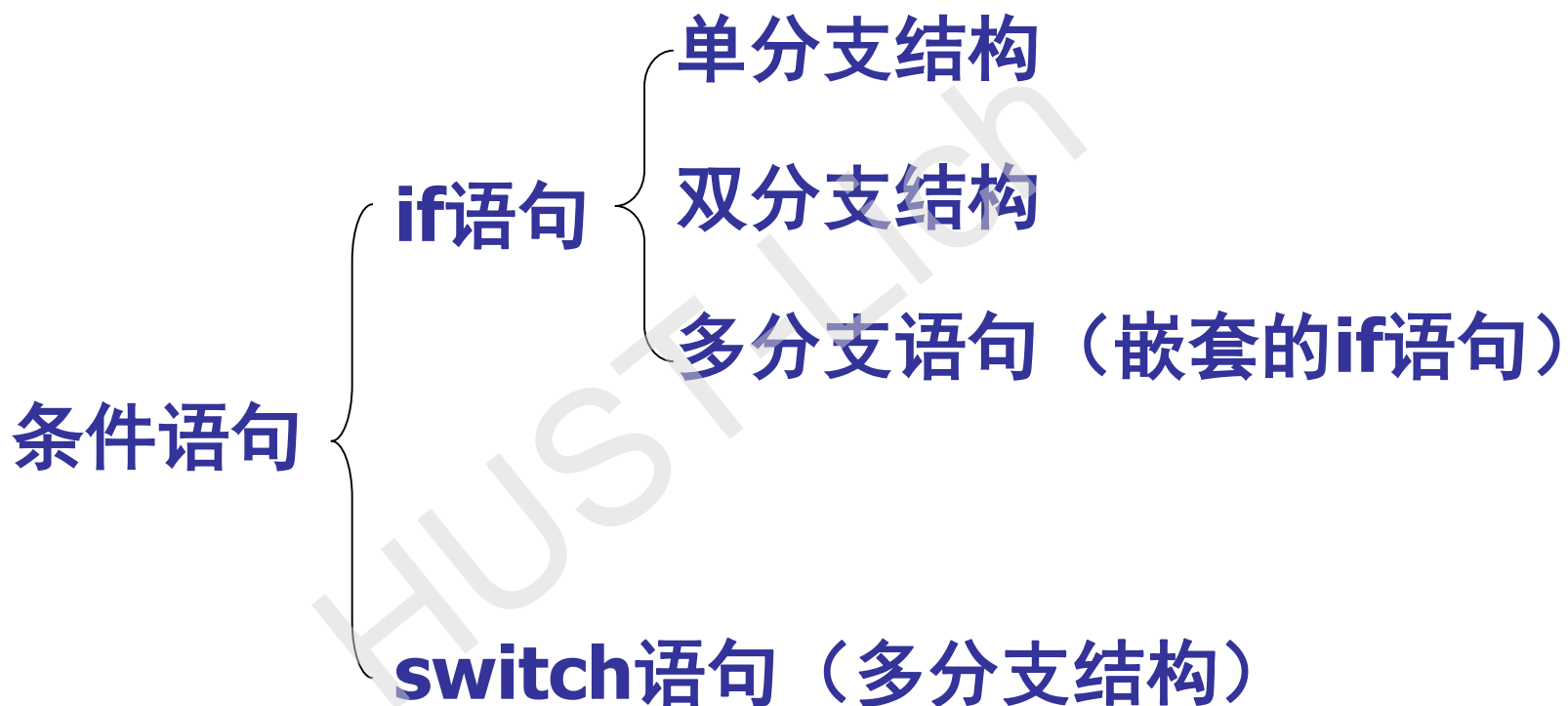
复合语句

占用两个不同的内存空间

变量的作用域：从定义变量开始，到遇到的第一个“}”范围内
超出变量的作用域，该变量所占用的内存空间就被回收

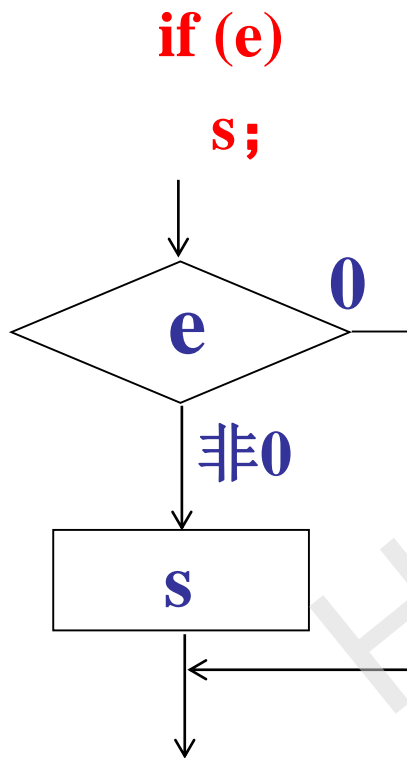


4.4 条件语句

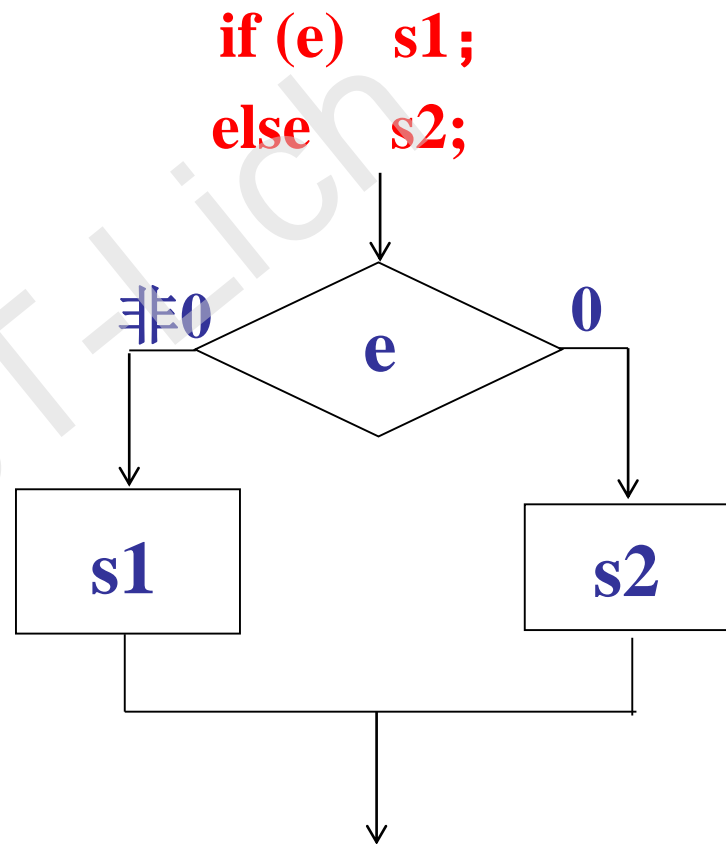


if 语句的一般形式

形式1：单分支结构



形式2：双分支结构



建议结构化语句的执行体部分都采用复合语句，清晰直观

输入10个数，统计正数、负数、零的个数

```
positive=negative=zero=0;
```

```
for(k=1;k<11;k++) {
```

```
    scanf("%d",&x);
```

```
    if(x>0) positive++;
```

```
    if(x<0) negative++;
```

```
    if(x==0) zero++;
```

```
    if(x>0) positive++;
```

```
    else if(x<0) negative++;
```

```
    else zero++;
```

```
    x>0 ? positive++ : (x<0 ? negative++ : zero++);
```

```
}
```

哪种效率最低？

哪种效率最高？

```
if(x>0) positive++;
```

```
else
```

```
{
```

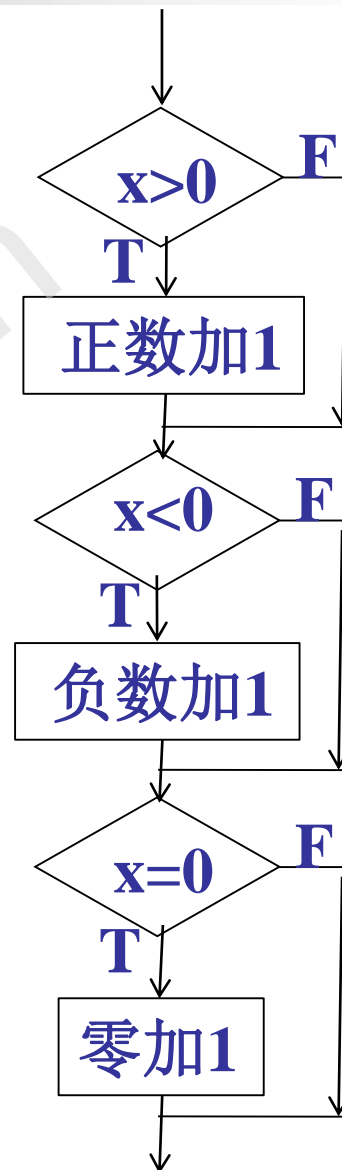
```
    if(x<0) negative++;
```

```
    else zero++;
```

```
}
```

利用多条单分支if实现多分支

```
//统计正数、负数、零的个数
positive=negative=zero=0;
for(k=1;k<11;k++) {
    scanf("%d",&x);
    if(x>0) positive++;
    if(x<0) negative++;
    if(x==0) zero++;
}
```



if语句实现多分支结构

//统计正数、负数、零的个数

positive=negative=zero=0;

for(k=1;k<11;k++) {

scanf("%d",&x);

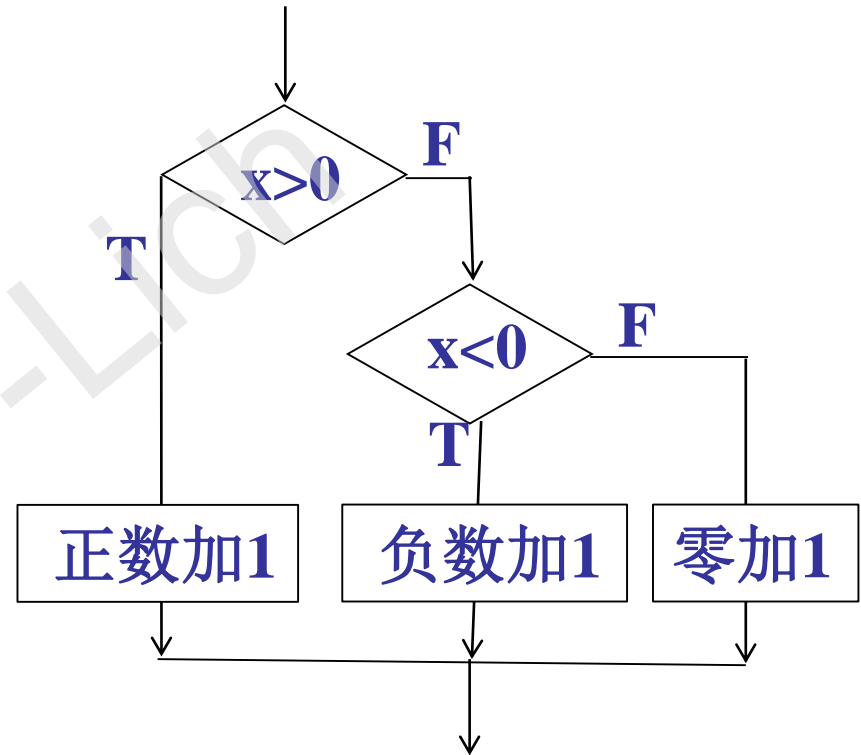
if(x>0) positive++;

else if(x<0) negative++;

else zero++;

}

x>0 ? positive++ : (x<0 ? negative++ : zero++);



嵌套的if语句实现多分支结构

//统计正数、负数、零的个数

positive=negative=zero=0;

for(k=1;k<11;k++) {

scanf("%d",&x);

if(x>0) positive++;

else if(x<0) negative++;

else zero++;

}

if(x>0) positive++;

else

{

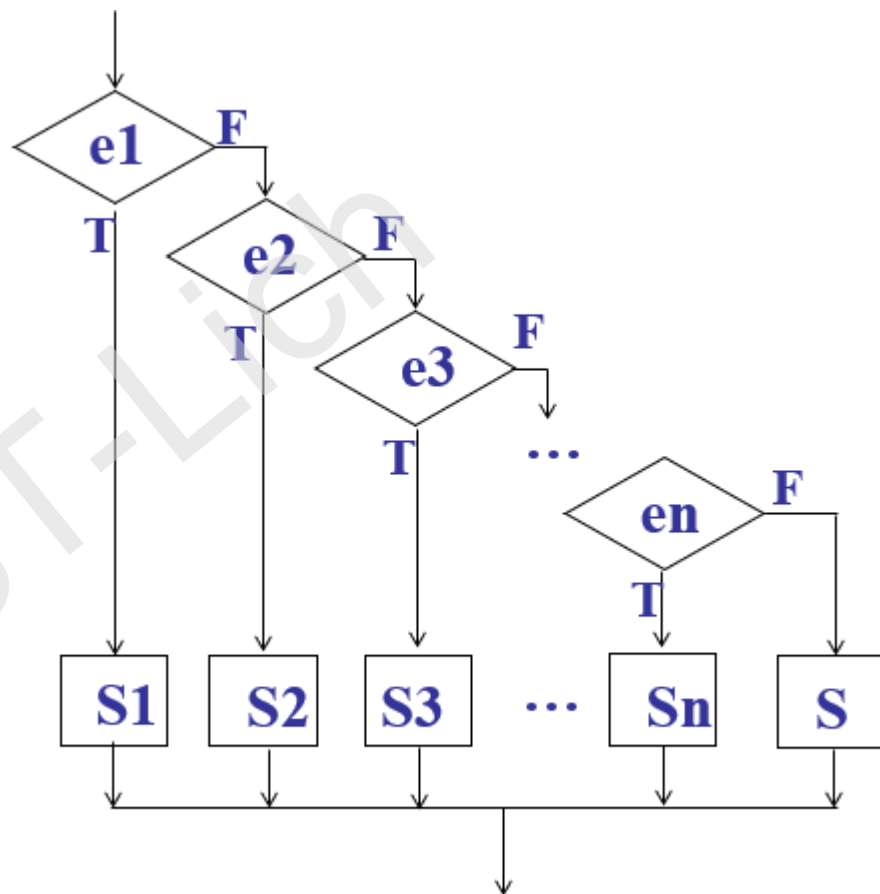
if(x<0) negative++;

else zero++;

}

else-if 语句实现多分支结构

```
if(e1)
    S1
else if (e2)
    S2
else if (e3)
    S3
...
else if(en)
    Sn
else S
```



各表达式依次求值，一旦某个表达式值为真，则执行其后面语句，并终止整个语句序列的执行。

嵌套的if语句

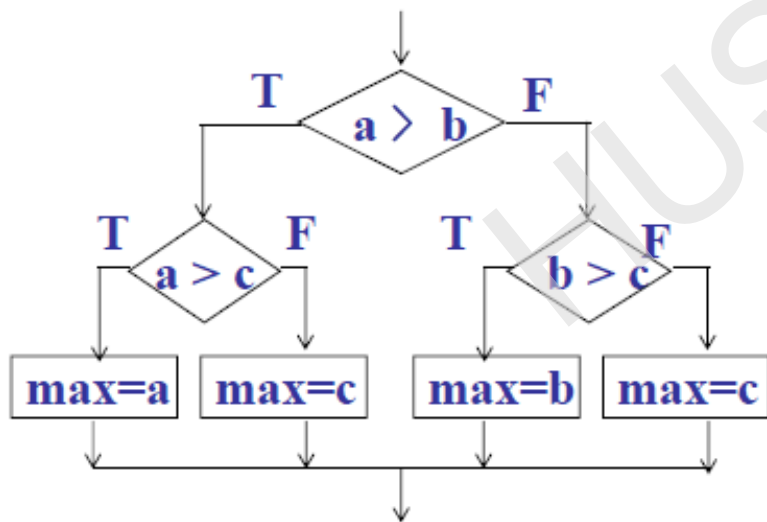
用嵌套的if语句求a, b, c三个数中最大值。

if (e)

if 语句

else

if 语句



if (a > b)

if (a > c) max = a;

else max = c;

else

if (b > c) max = b;

else max = c;

利用三目运算符(?:)

max=(a>b) ? a : b;

max=(max>c) ? max : c;

三个数的最大值

下面代码能实现求a, b, c中最大值吗?

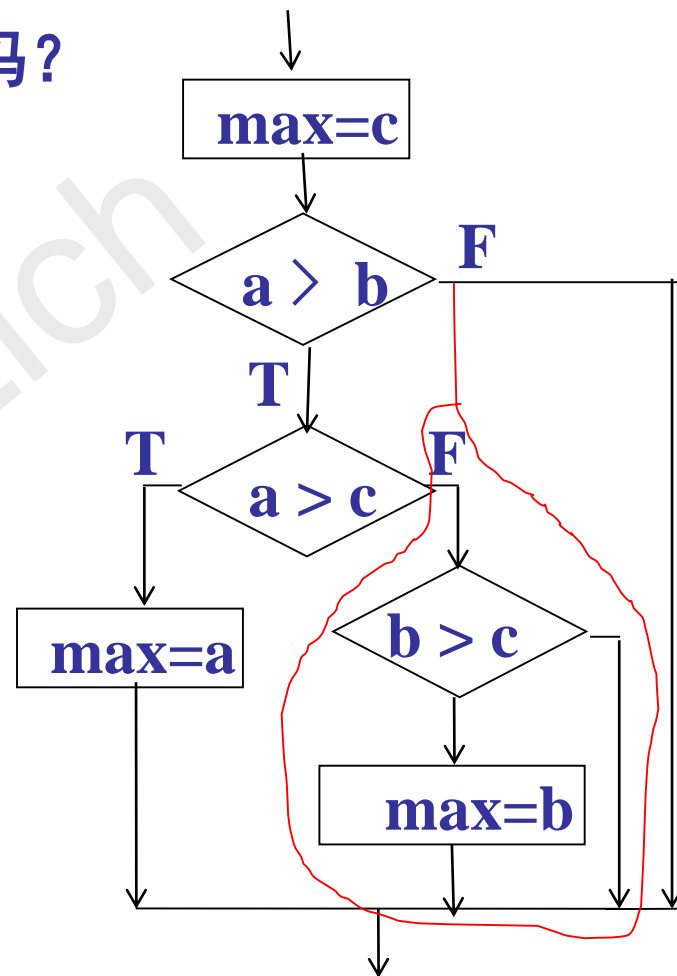
```
max = c;
```

```
if ( a > b )
```

```
    if ( a > c ) max = a;
```

```
    else
```

```
        if ( b > c ) max = b;
```



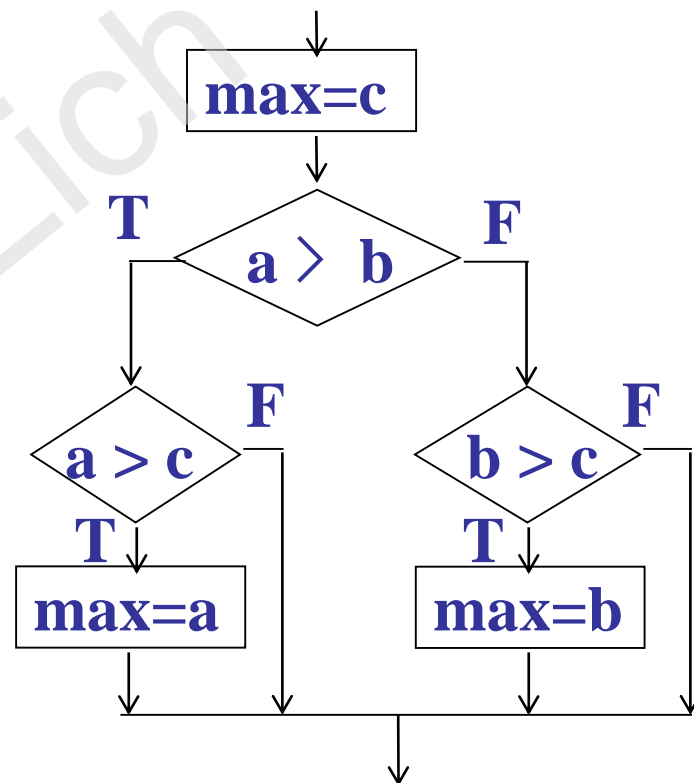
else配对规则: 内层优先配对原则

else与其同一程序块中最靠近的还未配对的if配对。

else的配对规则

else与其同一程序块中最靠近的还未配对的if配对。

```
max = c;  
if ( a > b ) {  
    if ( a > c ) max = a;  
}  
else  
    if ( b > c ) max = b;
```



为避免理解错误，增强程序可读性，在嵌套的if语句中最好将if子句和else子句分别用{ }括起来



程序设计举例【例4.2】

输入百分制成绩 x ，输出英文等级。

分数范围

$90 \leq x \leq 100$

$80 \leq x < 90$

$60 \leq x < 80$

$x < 60$

等级英文名

excellent (优)

good (良)

middle (中)

bad (差)

方案1—嵌套的if多分支结构

```
#include<stdio.h>
int main (void)
{
    float x;
    printf("input the score\n");
    scanf("%f", &x);           // 输入double: %lf
    if ( x > 100 || x < 0) printf("input error!\n");
    else {
        if (x>=90) printf(" excellent! \n");
        else if (x>=80) printf(" good! \n");
        else if (x>=60) printf(" middle! \n");
        else printf(" bad \n");
    }
    return 0;
}
```

方案2--多条单分支if

```
#include<stdio.h>
int main (void)
{
    float x;
    printf("input the score \n ");
    scanf("%f ", &x);
    if ( x > 100 || x < 0) printf("input error!\n");
    else {
        if (x>=90) printf(" excellent! \n");
        if (x<90&&x>=80) printf(" good! \n");
        if (x<80&&x>=60) printf(" middle! \n");
        if(x<60) printf(" bad \n");
    }
    return 0;
}
```

必有

方案3--switch语句

```
float x;  
printf("input the score x\n");  
scanf("%f ", &x);  
if ( x > 100 || x < 0)   printf("input error!\n");  
else
```

```
switch ( (int)(x/10) )
```

```
{
```

```
case 10:
```

```
case 9: printf(" excellent! \n"); break;
```

```
case 8: printf(" good! \n"); break;
```

```
case 7:
```

```
case 6: printf(" middle! \n"); break;
```

```
default: printf(" bad! \n"); break;
```

```
}
```

空的case语句，用于
几种情况合并执行一
组语句。

switch 语句——多分支语句

表达式 **e** 的值必须为一整型 (整型、字符型、枚举型均可)

一般形式:

```
switch (e) {  
    case 常量1: 语句序列1  
                break;  
    case 常量2: 语句序列2  
                break;  
    ...  
    case 常量n: 语句序列n  
                break;  
    default   : 语句序列  
                break;  
}
```

(又称开关语句)

switch语句执行流程

计算 **e**,
若 **e = 常量i**, 则执行其后的语句,直到 break 语句止;
若 **e** 与所有常量值不相等,则执行 **default** 子句;
若无 **default**, 则不执行switch中的任何语句。

break 用于跳出switch语句

switch 语句

下面语句执行时的输出？

```
a = 0;
```

```
for(i=1;i<=3;i++) {
```

```
    switch ( i ){
```

```
        case 0: a++;
```

```
        case 1: a++;
```

```
        case 2: a++;
```

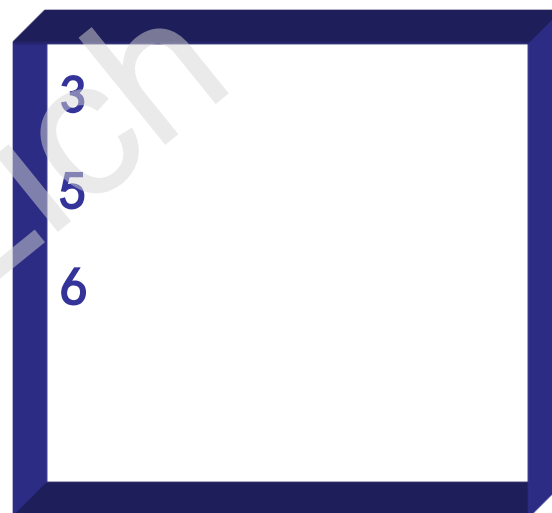
```
        default: a++;
```

```
    }
```

```
    printf("%d\n", a);
```

```
}
```

case语句后无break; 程序依次执行后面的语句，
包括default语句



方案3--switch语句

```
float x;  
printf("input the score x\n");  
scanf("%f ", &x);  
if ( x > 100 || x < 0)   printf("input error!\n");  
else  
    switch ( (int)(x/10) )  
    {  
        case 10:  
        case 9: printf(" excellent! \n"); break;  
        case 8: printf(" good! \n"); break;  
        case 7:  
        case 6: printf(" middle! \n"); break;  
        default: printf(" bad! \n"); break;  
    }
```

空的case语句，用于
几种情况合并执行一
组语句。



注意

当多个case执行相同的语句时，这些case可以写成一行。

case 7: case 6: printf(" middle! \n"); break; ✓

case 7, case 6: printf(" middle! \n"); break; ✗

case 7, 6: printf(" middle! \n"); break; ✗

建议还是竖排方式，清晰直观！

case 7:

case 6: printf("middle!\n"); break;



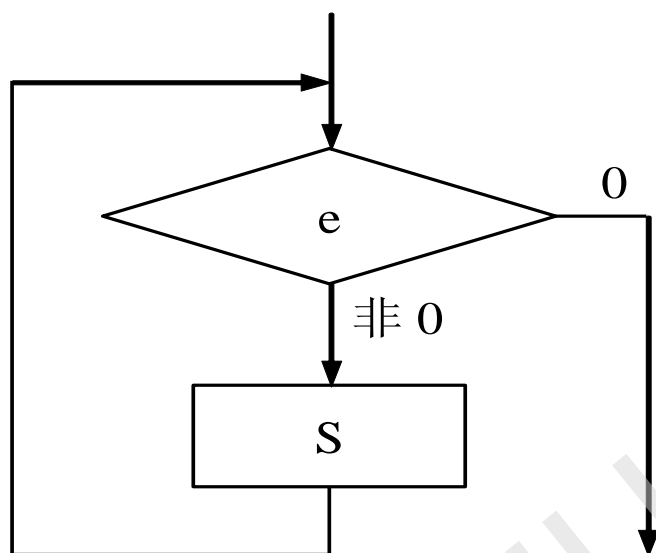
4.5 循环语句

循环结构是满足给定条件时，反复执行某个程序段。
反复执行的程序块叫循环体。

循环语句 {
 while语句
 for语句
 do while语句

4.5.1 while语句

认真阅读 例4.5-4.7



循环控制
表达式

while (e)

S

循环体
语句

i 为循环变量

```
while (i < 100)
{
    sum += i;
    i++;
}
```

循环语句应有出口，否则进入死循环

表达式中一般至少包括一个能够改变表达式值的变量，
这个变量称为**循环变量**

例

用 while 循环语句编写：求1到100之间所有整数之和的程序

```
#include <stdio.h>
```

```
int main( )
```

```
{
```

```
    int i=1 ;
```

```
    int sum =0;
```

定义变量，并赋初值

```
    while ( i <= 100)
```

```
    {
```

```
        sum += i ;
```

```
        i ++;
```

```
    }
```

```
    printf("sum = %d\n" , sum);
```

```
    return 0;
```

```
}
```

当 i>100时
跳出循环

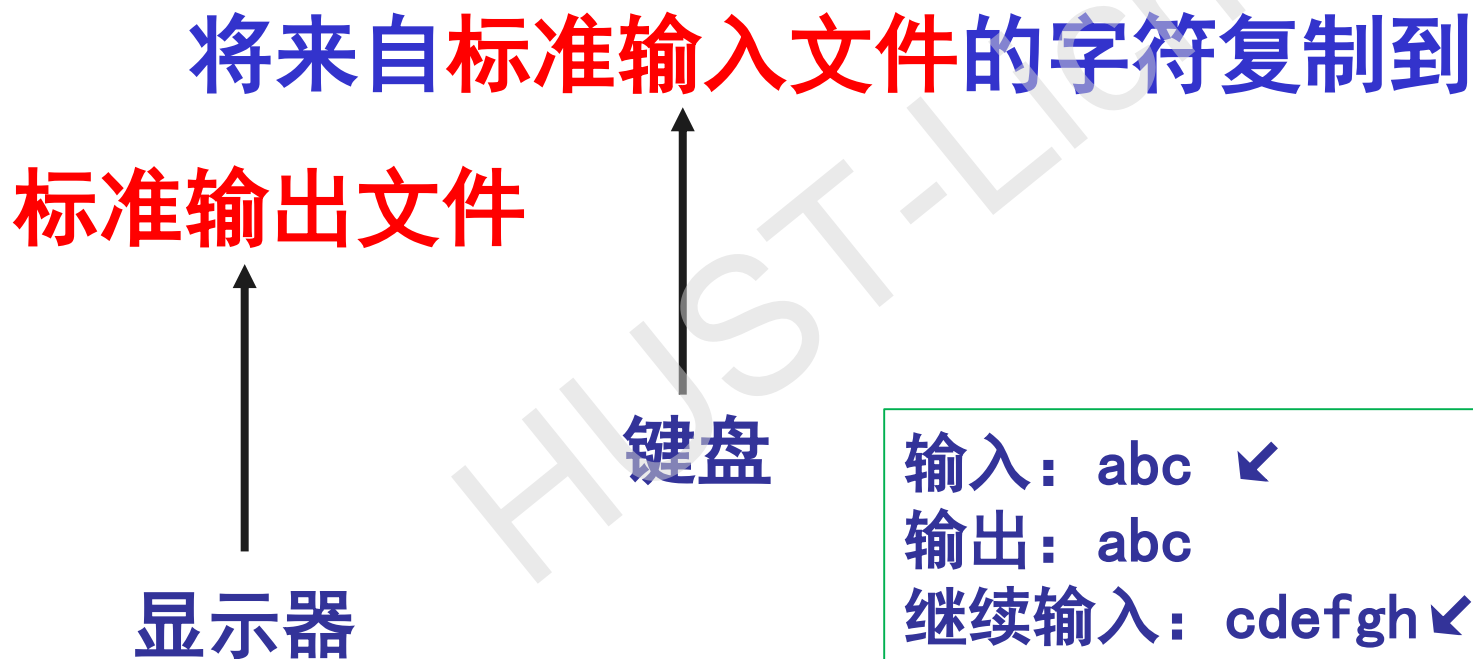
运行结果

sum =5050

若求1~100之间的奇数和、偶数和，程序如何修改？

举例：将标准输入转换成标准输出

【例4.8】将来自标准输入文件的正文复制到标准输出文件，每次输入和复制一个字符。



输入：abc ✓

输出：abc

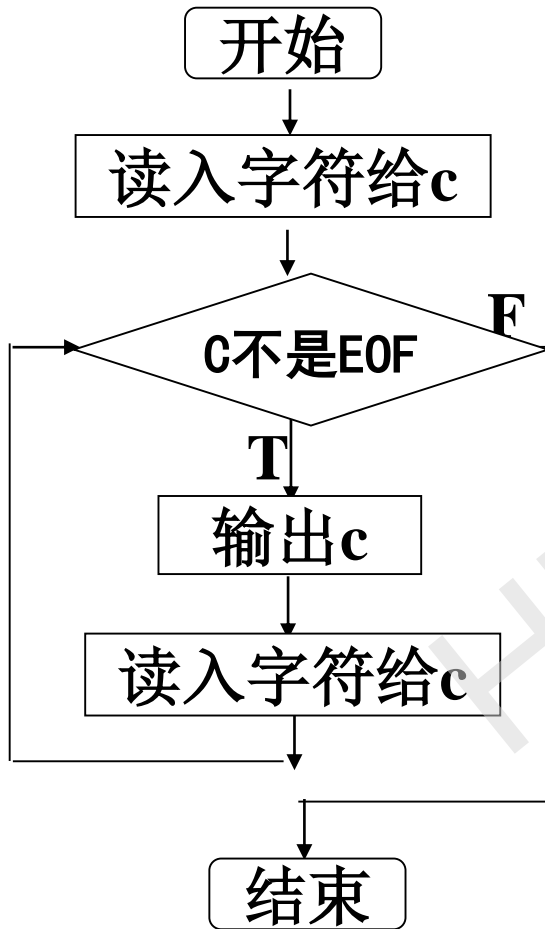
继续输入：cdefgh ✓

输出：cdefgh

.....

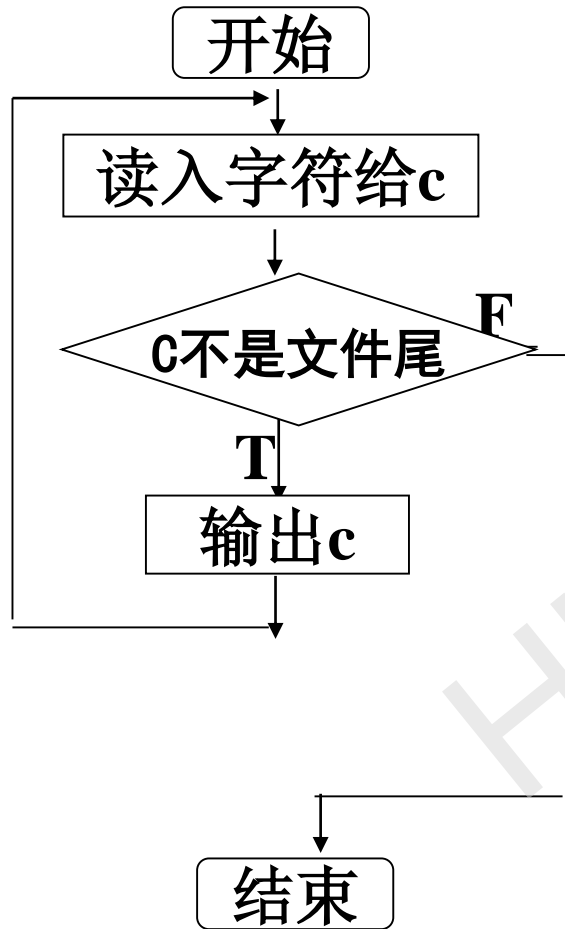
直到输入：Ctrl+z ✓ 结束输入

流程图与程序



```
#include<stdio.h>
int main( void )
{
    char c;
    printf( “ Input characters:\n” );
    c=getchar( );
    while(c != EOF) {
        putchar(c) ;
        c=getchar( );
    }
    return 0;
}
```


简化程序



```
#include<stdio.h>
int main(void)
{
    char c;
    printf( " Input characters :\n" );
    while( (c=getchar()) !=EOF )
        putchar(c);
    return 0;
}
```

举例：删除正文前面空格

将输入正文每一行的前导空格删除后，将后面部分进行输出

```
int main()
{
    // 方法1 03 金珂羽
    int flag=1; //前导空格标志
    char c;

    while((c=getchar())!=EOF)
    {
        if(c==' ' && flag==1) continue;
        else if(c=='\n') flag=1;
        else flag=0;

        putchar(c);
    }
    return 0;
}
```

```
int main()
{
    //方法3 04孙正阳 a
    int flag=0;
    char c;

    while((c=getchar())!=EOF)
    {
        if(!flag){
            if(c!=' ')
                flag=1;
        }
        if(flag) putchar(c);
        if(c=='\n') flag=0;
    }
    return 0;
}
```

举例：删除正文前面空格

将输入正文每一行的前导空格删除后，将后面部分进行输出

分析：用有限状态机（FSM—Finite State Machine）方法。状态机的4个要素：**现态、条件(事件)、动作、次态**

一句话描述：

当系统处于某**状态**时，
如果发生了什么**事件**，
就执行某**动作**，
然后系统变成**新状态**

```
while ( 事件 )  
    switch (当前状态) {  
        case 状态i:  
            if (某条件/事件)  
                执行动作, 迁移至状态j  
            break;  
        case 状态j:  
            .....  
            .....  
    }
```



FSM的4要素

现态：当前所处的状态

条件（事件）：当一个条件被满足，将会触发一个动作，或者执行一次状态的迁移。

动作：条件满足后执行的动作，动作不是必需的，当条件满足后，也可以不执行任何动作，直接迁移到新状态。也可以仍旧保持原状态。

次态：条件满足后要迁往的新状态，“次态”一旦被激活，就转变成新的“现态”了。

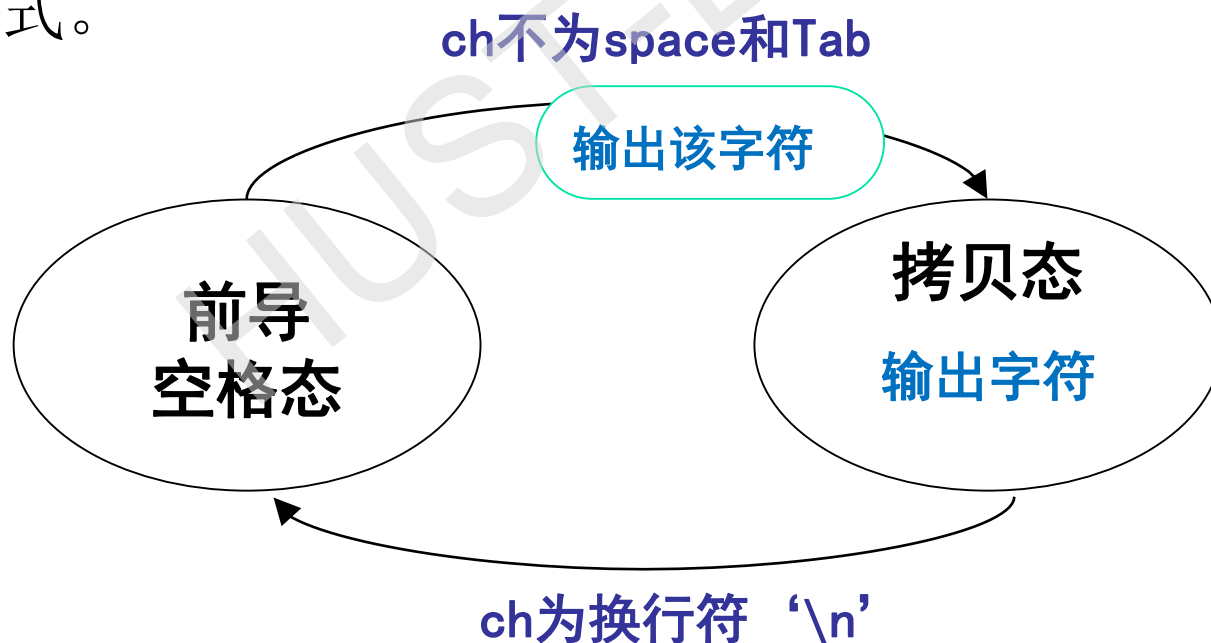
状态迁移图

使用状态机解决问题，主要有两个步骤：

(1) 确定系统总共有几个状态

(2) 确定状态之间的迁移过程

状态迁移图是一种描述系统的状态、以及状态相互转化关系的图形方式。



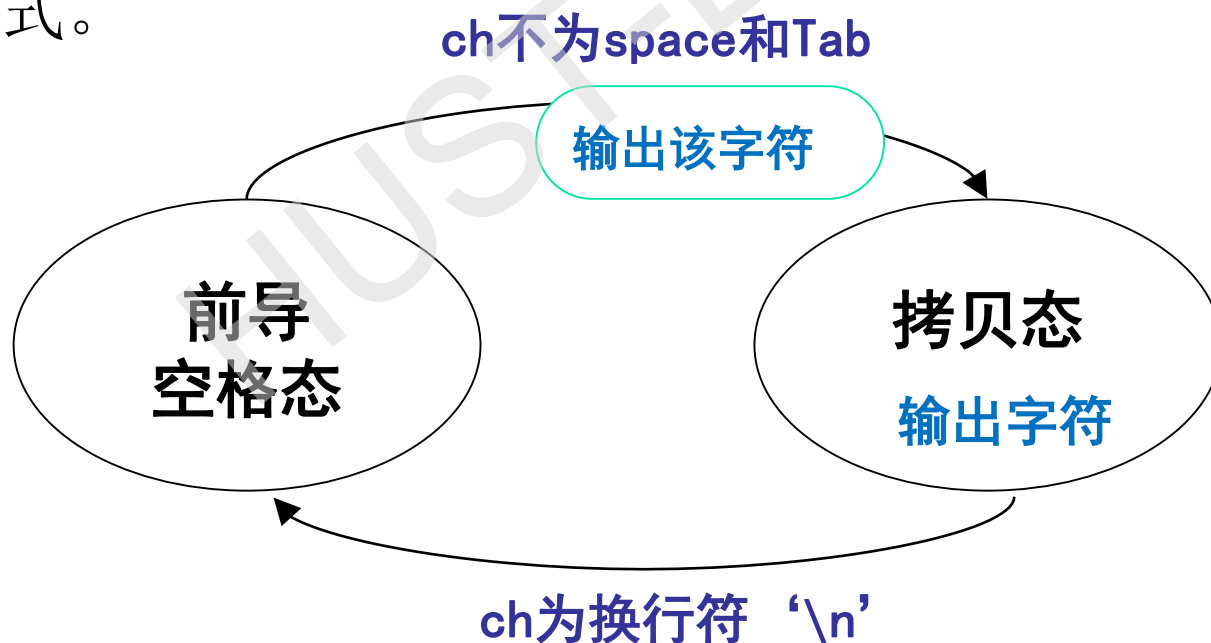
状态迁移图

使用状态机解决问题，主要有两个步骤：

(1) 确定系统总共有几个状态

(2) 确定状态之间的迁移过程

状态迁移图是一种描述系统的状态、以及状态相互转化关系的图形方式。



程序框架

```
while ((ch = getchar()) != EOF)
{
```

```
    switch(当前状态) {
```

```
        case 前导空格:
```

```
            if (ch不为空白符)
```

```
                输出ch;
```

```
                状态迁移至“拷贝态”
```

```
            break;
```

```
        case 拷贝 :
```

```
            输出ch;
```

```
            if (ch为换行符)
```

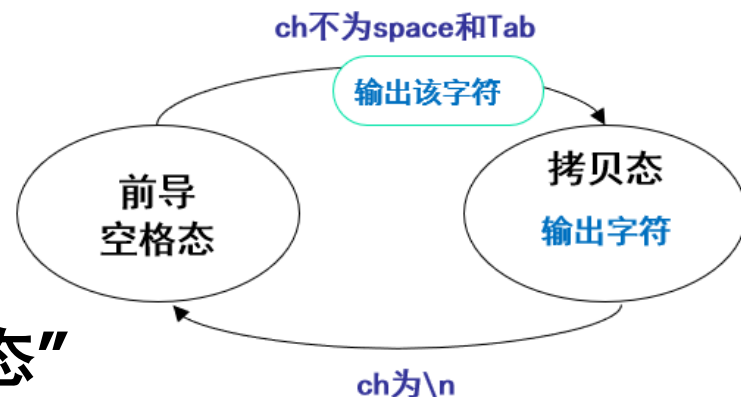
```
                状态迁移至“前导空格态”
```

```
            break;
```

```
    }
```

```
}
```

```
while ( 事件 )
switch (当前状态) {
    case 状态i:
        if (某条件/事件)
            执行动作, 迁移至状态j
        break;
    case 状态j:
        .....
        .....
}
```



```
enum{HeadSpace,COPY};
```

```
int main()
```

```
{
```

```
    int ch,state;    //state存当前状态
```

```
    state=HeadSpace; //初始前导空格状态
```

```
    printf("input text, end of Ctrl+z\n");
```

```
    while ((ch = getchar()) != EOF)
```

```
    {
```

```
        switch(state) {
```

```
            case HeadSpace:
```

```
                if (ch!=' ' && ch!='\t') //不为空格时
```

```
                { putchar(ch); state=COPY; }
```

```
                break;
```

```
            case COPY:
```

```
                putchar(ch);
```

```
                if(ch=='\n')
```

```
                state=HeadSpace;
```

```
                break;
```

```
        }
```

```
    }
```

```
    return 0;
```

```
}
```

```
while ((ch = getchar()) != EOF)
```

```
{
```

```
    switch(当前状态) {
```

```
        case 前导空格:
```

```
            if (ch不为空白符)
```

```
                输出ch;
```

```
                状态迁移至“拷贝态”
```

```
            break;
```

```
        case 拷贝 :
```

```
            输出ch;
```

```
            if(ch为换行符)
```

```
                状态迁移至“前导空格态”
```

```
            break;
```

```
    }
```

```
}
```




举例：删除注释

【例4.10】 输入一个C程序(一段正文)，按原来格式复制输出，复制过程中删去所有的注释（标准C的注释）。

分析：为了删去C程序中所有的注释，关键在于如何区分注释部分和需要复制的部分。为此，可将复制过程划分为4种状态：

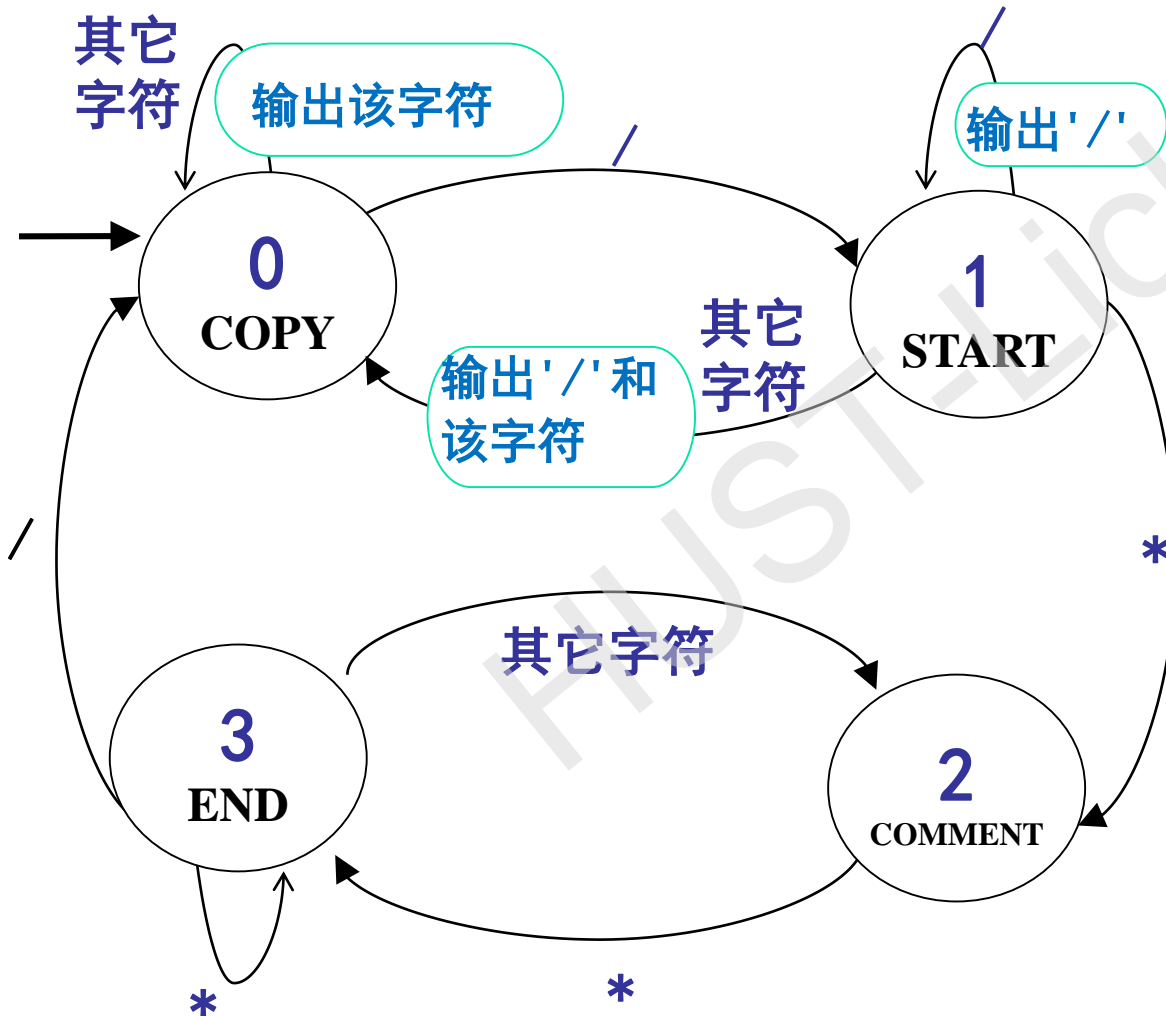
0--复制状态(COPY)： 初始状态

1--开始注释状态 (START)

2--注释状态 (COMMENT)

3--结束注释状态 (END);

状态迁移图



```
case START: /* 在START处 */
    if (c == '*') /* 如果 */
        state = COMMENT;
    else if (c == '/') { /*
        putchar('/');
    }
    else { /* 如果 */
        putchar('/');
        putchar(c);
        state = COPY;
    }
    break;
```



自主练习

1、修改例4.10，使之能处理字符串常量。

`“/* comment in\” string */”`

2、使用状态机方法统计单词个数

输入一段英文，统计单词数。单词仅包含大写字母和小写字母，单词之间用空白符或标点符号隔开。

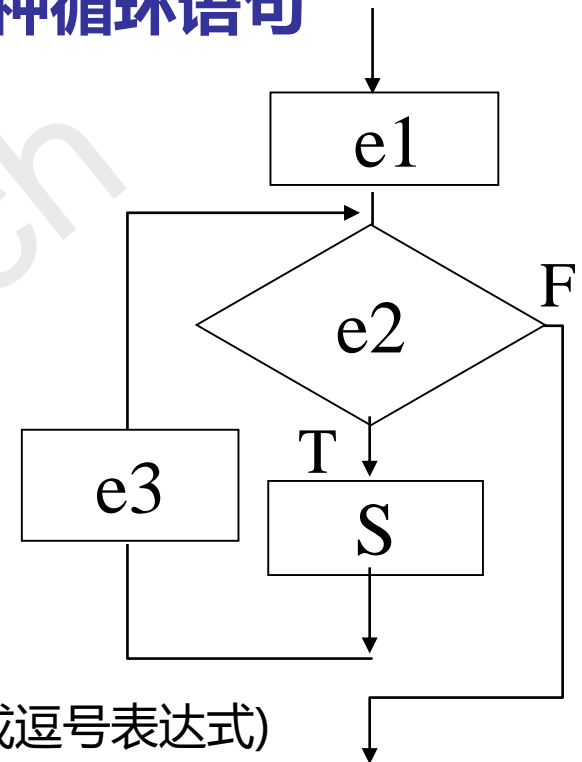
4.5.3 for 语句

for 语句是功能最强、使用最灵活的一种循环语句

for 语句结构更简洁

for(e1; e2; e3)
S

e1;
while(e2) {
S
e3;
}



e1, 给循环变量或相关变量赋**初始值** (赋值表达式或逗号表达式)

e1仅执行一次

e2, **循环条件** (关系表达式或逻辑表达式)

e3, **修改循环变量的值** (赋值表达式、逗号表达式、自增自减表达式)

e3也可以执行循环体中的其他运算

例

用 for 循环实现求1到100之间所有整数之和

```
#include <stdio.h>
```

```
int main( )
```

```
{
```

```
    int i ;
```

```
    int sum ;
```

表达式一，为逗号表达式，
赋初值

- for (i = 1 , sum = 0 ; i <= 100; i ++)

```
{
```

```
    sum += i ;
```

表达式二：
循环条件

表达式三：
循环步长

```
}
```

```
    printf("sum = %d \n", sum);
```

```
    return 0;
```

```
}
```

```
int main( )
```

```
{
```

```
    int i=1 ;
```

```
    int sum =0;
```

- while (i <= 100)

```
{
```

```
    sum += i ;
```

```
    i ++;
```

```
}
```

```
    printf("sum = %d\n" , sum);
```

```
    return 0;
```

```
}
```

```
for (i = 1 , sum = 0 ; i <= 100; sum+=i, i ++)  
    ; //空语句
```

for语句的几种特例

- 表达式1、2、3可以全部或部分省掉，但是**分号不能省**



只有e2时

for(; e2 ;) S \longleftrightarrow while (e2) S



缺省e2时

for(e1; ;e3) S \longleftrightarrow for(e1; 1 ; e3) s



e1,e2,e3全为空

for(; ;) S \longleftrightarrow while(1) S

必须在循环体中使用**break**终止循环;
否则进入死循环

- 循环体也可以为空语句，如:

for(i=0 ; i<10000 ; i++) ;

或 for(i=0 ; i<10000 ; i++) { }

作用：延迟一段时间

举例：最值问题

输入一批正整数，以0为结束，输出其中最大值。

```
#include<stdio.h>
int main(void)
{
    int x, max;
    printf("input a group integer end of 0: \n");
    scanf("%d", &x);
    max = x; /* 初始以第一个数为擂主 */
while(x) for( ; x ; ) { /* x 等价于 x!=0 */
        scanf("%d", &x);
        if (max < x) max = x;
    }
    printf("max=%d\n", max);
    return 0;
}
```



输入一批正整数，以**ctrl+z**为结束，输出其中最大值。

```
#include<stdio.h>
```

```
int main(void)
```

```
{
```

```
    int x, max;
```

```
    max=1<<sizeof(int)*8-1; // int的最小数作为擂主
```

```
    printf("input a group integer end of ctrl+z: \n");
```

```
    while(scanf("%d", &x) !=EOF)
```

```
        if (x > max) max = x;
```

```
    printf("max=%d\n", max);
```

```
    return 0;
```

```
}
```




输入一批正整数，以**ctrl+z**为结束，输出其中最大值。

```
#include<stdio.h>
```

```
int main(void)
```

```
{
```

```
    int x, max;
```

```
    max=1<<sizeof(int)*8-1; // int的最小数作为擂主
```

```
    printf("input a group integer end of ctrl+z: \n");
```

```
    while(scanf("%d", &x)==1)
```

```
        if (x > max) max = x;
```

```
    printf("max=%d\n", max);
```

```
    return 0;
```

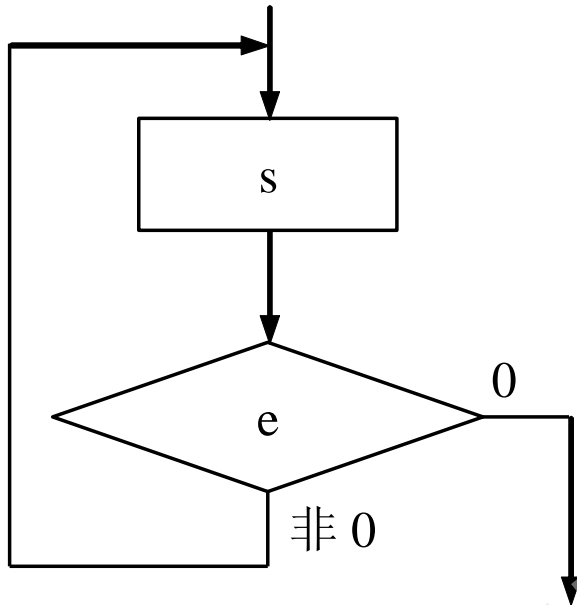
```
}
```

举例：求阶乘 $n!$

求 $n!$ ， n 从终端输入。

```
#include<stdio.h>
int main(void)
{
    int n, i;
    unsigned long long fac; // C99标准
    printf("input n:\n");
    scanf("%d", &n);
    for (fac = 1, i = 1; i <= n; i++)
        fac *= i;
    printf("%d! = %llu\n", n, fac); // 或者 %l64u
    return 0;
}
```

4.5.2 do-while语句



后测试循环条件

do

S

while (e);

等价于

S

while (e)

S

如:

```
int i=0;
```

```
do
```

```
{
```

```
    printf("%3d" , i++);
```

```
}while(i<5);
```

0 1 2 3 4

能用do-while循环描述的程序一定能用while循环和for循环描述，
反之不一定

例

do-while实现: 求1到100之间所有整数之和

```
#include <stdio.h>
void main( )
```

```
{
    int i=1;
    int sum=0;
```

- do

```
{
```

```
    sum += i;
    i ++;
```

当 i>100时
跳出循环

- ```
 } while (i<=100);
 printf("sum = %d \n", sum);
}
```

**for 语句结构更简洁**

```
int main()
{
 int i;
 int sum;
```

- ```
    for (i = 1 , sum = 0 ; i <= 100; i ++)
```

```
    {
        sum += i;
    }
    printf("sum = %d \n", sum);
    return 0;
}
```

```
int main( )
```

```
{
```

```
    int i=1;
    int sum =0;
```

- ```
 while (i <= 100)
 {
 sum += i;
 i ++;
 }
```

```
 printf("sum = %d\n" , sum);
 return 0;
```

```
}
```



# 举例：反向输出

**【例4.11】** 把输入的正整数按反方向输出。

例如，输入的数是12345，要求输出结果是54321。

**分析：** 输入：从高位到低位。

输出：从低位到高位

**关键：** 如何获取一个整数（ $x$ ）的个位数字。

**算法：** (1)  $x \% 10$

(2)  $x = x / 10$ ，不停循环语句直至  $x = 0$



# 程序

```
/*把输入的正整数按反方向输出*/
#include<stdio.h>
int main(void)
{
 int x, digit;
 printf("input an integer x>0:\n");
 scanf("%d", &x);
 do {
 digit = x % 10; /* 求个位数字 */
 printf("%d", digit);
 x /= 10; /* 十位数字变个位数字 */
 }while (x);
 return 0;
}
```



# 程序

```
/*把输入的整数按反方向输出*/
#include<stdio.h>
int main(void)
{
 int x, digit;
 printf("input an integer:\n");
 scanf("%d", &x);
 while (x) {
 digit = x % 10; /* 求个位数字 */
 printf("%d", digit);
 x /= 10; /* 十位数字变个位数字 */
 }
 return 0;
}
```

# 菜单程序框架

```
1: 输入成绩
2: 找最高分
3: 统计各分数段人数
```

```
3
3: 各分数段人数....
```

```
continue ? yes or no?
```

```
y
```

```
1: 输入成绩
2: 找最高分
3: 统计各分数段人数
```

```
2
```

```
2: 最高分....
```

```
continue ? yes or no?
```

```
y
```

```
1: 输入成绩
2: 找最高分
3: 统计各分数段人数
```

```
1
```

```
1: 输入成绩....
```

```
continue ? yes or no?
```

```
n
```

```

Process exited after 94.41 seconds with return value 0
请按任意键继续. . .
```





```
int main(void)
{
 char ch;
 //lop: /* 标号语句 */
 do{
 do { /* 构造提示菜单, 供用户选择 */
 printf("\t 1: 输入成绩\n");
 printf("\t 2: 找最高分 \n");
 printf("\t 3: 统计各分数段人数\n");
 scanf("%c%c",&ch); /* ch=getchar(); getchar(); */
 } while(ch < '1' || ch > '3');

 switch(ch) { /* 根据用户输入的选择转移 */
 case '1':
 printf("\t 1: 输入成绩....\n");
 break;
 case '2':
 printf("\t 2: 最高分....\n");
 break;
 case '3':
 printf("\t 3: 各分数段人数....\n");
 break;
 }
 printf("continue ? yes or no?\n");
 ch = getchar(); getchar(); /* 是否继续处理 */
 // if(ch == 'y' || ch == 'Y') goto lop; /* goto语句 */
 }while(ch == 'y' || ch == 'Y') ;
 return 0;
}
```



```
int main(void)
{
 char ch;
lop: /* 标号语句 */
//do{
do { /* 构造提示菜单, 供用户选择 */
 printf("\t 1: 输入成绩\n");
 printf("\t 2: 找最高分 \n");
 printf("\t 3: 统计各分数段人数\n");
 scanf("%c%c",&ch); /* ch=getchar(); getchar(); */
} while(ch < '1' || ch > '3');

switch(ch) { /* 根据用户输入的选择转移 */
 case '1':
 printf("\t 1: 输入成绩....\n");
 break;
 case '2':
 printf("\t 2: 最高分....\n");
 break;
 case '3':
 printf("\t 3: 各分数段人数....\n");
 break;
}
printf("continue ? yes or no?\n");
ch = getchar(); getchar(); /* 是否继续处理 */
if(ch == 'y' || ch == 'Y') goto lop; /* goto语句 */
//}while(ch == 'y' || ch == 'Y') ;
return 0;
}
```



# 转移语句

---

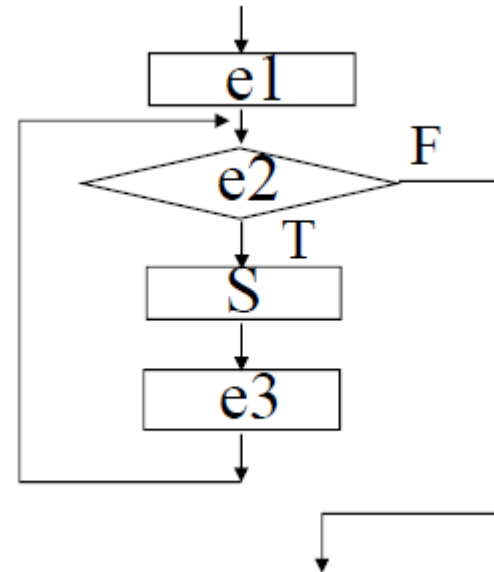
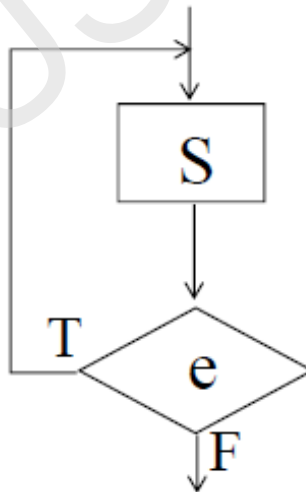
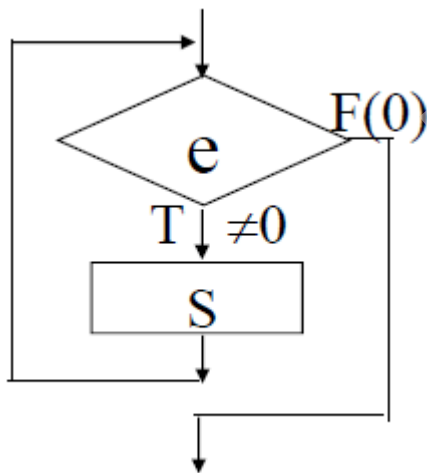
转移语句 {  
goto语句  
标号语句  
break语句  
continue语句

# break语句

只能用于循环体语句和switch语句：

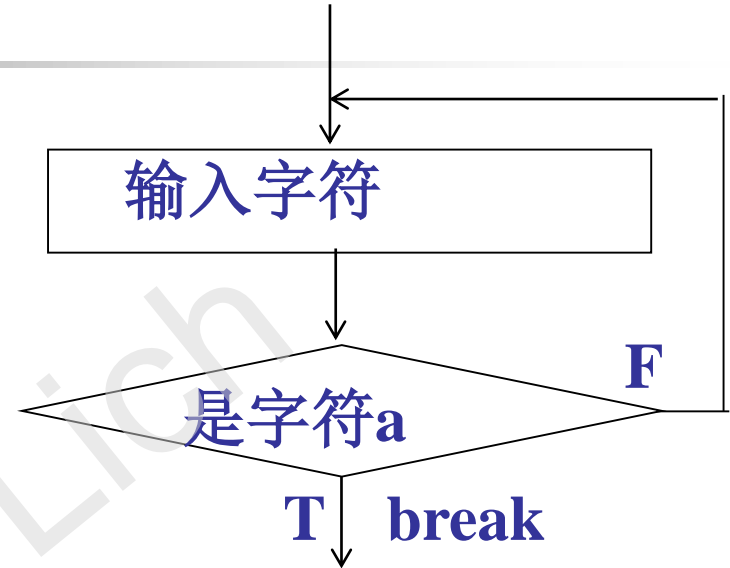
- 在switch语句中，用于**终止case子句的执行，跳出switch结构**。
- 用在循环体内部，强迫**立即终止循环**，跳过循环条件检测，**继续执行循环体下面的语句**。

注意：**break语句只能跳出当前一层循环。**



# 举例

```
for(;;) {
 ch = getchar () ;
 if (ch == 'a') break ;
}
```



改写

```
while ((ch = getchar ()) != 'a')
 ;
```

```
do
 ch = getchar () ;
while (ch != 'a') ;
```

```
for(ch = getchar () ; ch != 'a'; ch = getchar ())
 ;
```

# continue语句

- **功能：** 用在循环体内，跳过其后代码，继续下一轮循环。

/\*输出0 - 100之间能被5整除的数 \*/

```
int main()
{
 for(int n=0;n<=100;n++)
 {
 if(n%5)
 continue;
 printf("%d\t",n);
 }
 return 0;
}
```

if(!n%5)  
 printf("%d\t",n);

在某些场合使用  
continue语句可以提  
高整个程序的效率

程序功能？

# continue和break的区别

```
#include<stdio.h>
#include <math.h>
#define PI 3.1415926
int main()
{
 double r, area;
 while (1)
 {
 printf("input the radius:");
 scanf("%lf", &r);
 if (fabs(r) < 1e-6)
 break;
 else if (r < 0.0)
 {
 printf("the input is error\n");
 continue;
 }
 area = PI * r * r;
 printf("the area is:%lf\n", area);
 }
 return 0;
}
```

确定输入圆半径值有意义

输入一个圆的半径，输出圆的面积。

对这个程序进行改进，要求：

- (1) 允许反复的输入半径，计算并显示圆的面积，直到输入的半径是0为止；
- (2) 对输入的半径进行检查，若发现是负数将提示操作者重新输入。



**continue语句**只是结束本次循环，而不是中止整个循环，而**break语句**则是结束整个循环过程，不再判断执行循环的条件是否成立。

# goto 语句和标号语句

- 程序中使用goto语句时要求和标号配合，一般形式：

goto 标号;

.....

标号: 语句;

- **功能：** **无条件**把程序控制转移到标号指定的语句处。  
即执行goto语句之后，程序从指定标号处的语句继续执行。



# goto 语句和标号语句

用途:

(1) 与if语句一起构成循环语句

```
loop: if(e) {
 语句序列
 goto loop;
}
```

**为了使程序的结构化功能强,  
应尽量少用goto语句!**

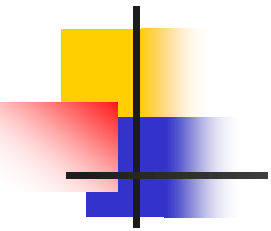
(2) 从循环体中跳转到循环体外

```
for(e1; e2; e3) {

 for(e1; e2; e3) {

 if(e) goto stop;
 }

}
stop: <.....
```



```
int main(void)
{ char ch;
 lop: // 标号语句
 do { // 构造提示菜单，供用户选择
 printf("\t 1: 输入成绩\n");
 printf("\t 2: 找最高分 \n");
 printf("\t 3: 统计各分数段人数\n");
 scanf("%c%c",&ch); // ch=getchar(); getchar();
 } while(ch < '1' || ch > '3');
 switch(ch) { // 根据用户输入的选择进入相应分支
 case '1': inputScore(x, n); break;
 case '2': maxScore(x, n); break;
 case '3': statistics(x, n); break;
 }
 printf("continue ? yes or no?\n");
 ch = getchar(); getchar(); /* 是否继续处理 */
 if(ch == 'y' || ch == 'Y') goto lop; /* goto语句 */
 return 0;
}
```

## 4.7.1

# 嵌套循环（多重循环）

- 先考虑while循环，在一重循环中， while循环语句的格式如下：

```
while(条件表达式)
{
 循环体部分;
}
```

- 当循环体部分的功能无法用顺序和选择结构而必须用循环结构来实现时，上述的结构就变成如下的形式：

```
while(条件表达式1)
{
 while(条件表达式2)
 {
 循环体部分2;
 }
}
```

外循环

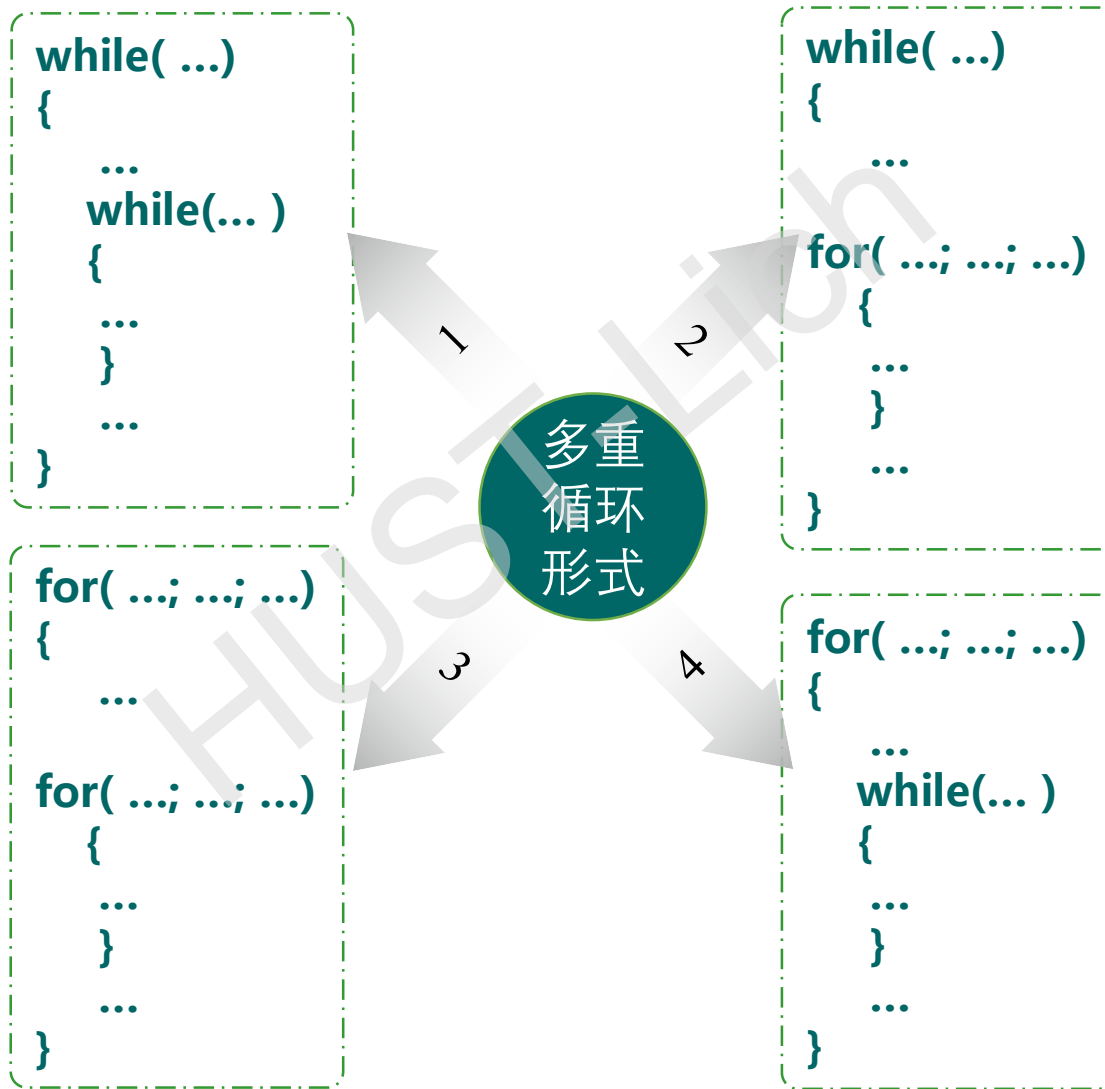
内循环

**break语句只能跳出当前层循环**

二重循环，可用来解决比单重循环更复杂的问题

# 多重循环

在多重循环中，内层循环可以认为是外层循环的循环体，依此类推。



```
for (i=1;i<=5;i++) {
 for(j=1;j<=i;j++)
 putchar('*');
 putchar('\n');
}
```

```
*
**


```

```
for (i=1;i<=5;i++)
 for(j=1;j<=i;j++)
 putchar('*');
 putchar('\n');
```

```

```

❗ 不要忘记内、外循环体中必须的 { }，否则将导致死循环或结果出错。

```
for (i=1;i<=5;i++) {
 for(j=1;j<=i;j++)
 putchar('*');
 putchar('\n');
}
```

```
*
**


```

```
for (i=1;i<=5;i++){
 for(j=1;j<=i;i++)
 putchar('*');
 putchar('\n');
}
```

循环输出 \* 进入死循环

❗ 内、外循环变量不应同名，否则将导致死循环或结果出错。

# 例

## 编程输出如下三角形图案

|       | 行号  | 空格数   | *数      |
|-------|-----|-------|---------|
| *     | 0   | 5     | 1       |
| ***   | 1   | 4     | 3       |
| ***** | 2   | 3     | 5       |
| ***** | 3   | 2     | 7       |
| ***** | 4   | 1     | 9       |
| ***** | 5   | 0     | 11      |
|       | $i$ | $5-i$ | $2*i+1$ |



首先找出每行 \*号前空格数, 建立行号与该行空格数、\*数的关系表达式

```
#include <stdio.h>
void main()
{
 int i, j; //分别定义行数、空格数、*数
 循环变量
 for (i = 0 ; i < 6 ; i ++)
 {
 printf("\n");
 for(j = 0 ; j < 5-i ; j ++)
 { //输出第 i 行左侧空格
 putchar(' ');
 }
 for(j=0 ; j<2*i+1 ; j++)
 { //输出第 i 行*号
 putchar('*');
 }
 }
}
```

## 二重循环举例

输入一个字母，在屏幕正中输出由这个字母决定其高度的字符"金字塔"。例如输入小写字母d，则输出下列左边图形，如果输入大写字母D，则输出右边图形。

```
 a
 a b a
 a b c b a
a b c d c b a
```

```
 A
 A B A
 A B C B A
A B C D C B A
```



## 分析：

```
(2) for(c1= top ; c1<=c; c1++) {
```

输出左侧字符：从top到c1的每个字符（递增）

输出右侧字符：从(c1-1)到top的每个字符（递减）

## 输出换行符

}

```
#include <stdio.h>
#include<ctype.h>
#define AXIS 40
```

```
int main()
```

```
{
 char c,c1,c2,top;
```

```
 int i, j;
```

```
 c=getchar();
```

```
 top=isupper(c)?'A':(islower(c)? 'a':'\0') ;
```

```
 if(top) { /* 如 top 非零，输出图形 */
```

```
 for(c1= top; c1<=c; c1++) {
```

```
 for (j=1; j<=2*(c-c1); j++) /* 靠左显示，输出左侧空格 */
```

```
 putchar(' ');
```

```
 for(c2=top;c2<=c1;c2++) /* 输出左侧，从top到c1的每个字符 */
```

```
 printf("%2c",c2);
```

```
 for(c2=c1-1;c2>=top;c2--) /* 输出右侧，从(c1-1)到top的每个字符 */
```

```
 printf("%2c",c2);
```

```
 printf("\n"); /* 换行 */
```

```
 }
```

```
 }
```

```
 return 0;
```

```
}
```

```

 a
 a b a
 a b c b a
a b c d c b a

```

```

 a
 a b a
 a b c b a
a b c d c b a

```

```
for(j=1;j<=AXIS-2*(c1-top);j++)
/* 居中显示*/
```

# 二重循环举例

求  $s = 1^1 + 2^2 + 3^3 + \cdots + n^n$  ,  $n$ 由终端输入。

分析:

(1) 输入 $n$ , 求和变量 $s$ 置初值0;

(2) `for(i=1;i<=n;i++) {`  
    `term = ii; for(term=1, j=1; j<=i; j++) term*= i;`  
    `s+=term;`  
    `}`

(3) 输出 $s$

# 二重循环举例1

```
int i, j, n;
long s, term;
printf ("Input n:");
scanf ("%d" , &n);
s=0;
for (i=1; i<=n; i++) {
 for(term=1, j=1; j<=i; j++) /* term=ii */
 term*=i;
 s+=term;
}
printf("s=%ld\n" , s);
```

◆ 注意给与循环有关的变量赋初值的位置。只需赋一次初值的操作应放在最外层循环开始执行之前，给内循环的有关变量赋初值应放在外循环体内、内循环开始执行之前

# 二重循环举例1

```
int i, j, n;
long s, term;
printf ("Input n:");
scanf ("%d" , &n);
s=0; term=1;
for (i=1; i<=n; i++) {
 for(j=1; j<=i; j++)
 term*=i;
 s+=term;
}
printf("s=%ld\n" , s);
```

◆ 会怎样?

//  $term = 1^1 * 2^2 * (i-1)^{i-1} * i^i$



# 二重循环举例1

```
int i, n;
long s;
printf ("Input n:");
scanf ("%d" , &n);
s=0;
for (i=1; i<=n; i++) {
 s+=power(i,i) ;
}
printf("s=%ld\n" , s);
```

```
//计算 x^n ($n>0$)
long power (int x, int n)
{
 int i;
 long p=1;
 for (i=1; i<=n; i++)
 p*=x;
 return p
}
```

1. 求  $s = 1^1 + 2^2 + 3^3 + \dots + n^n$ ，**n**由终端输入。

（提示：注意给与循环有关的变量赋初值的位置。只需赋一次初值的操作应放在最外层循环开始执行之前，给内循环的有关变量赋初值应放在外循环体内、内循环开始执行之前）

2. 上机实践：输出乘法九九表。

3. 输入一个字母，在屏幕正中输出由这个字母决定其高度的字符“金字塔”。例如输入小写字母**d**，则输出下列左边图形，如果输入大写字母**D**，则输出右边图形。（提示：函数isupper(c)、islower(c)用于判断字符c是否大写或小写，若是返回1；否则0。这两个函数在头文件ctype.h中）

```

 a
 a b a
 a b c b a
a b c d c b a

```

```

 A
 A B A
 A B C B A
A B C D C B A

```

本节将介绍三种处理简单问题的程序设计典型模板。

- 多项式累计法
- 显式条件试数法
- 隐式条件处理法

本章我们强调了程序的风格和结构的规范化，但是当我们面对一个较为复杂的编程问题时，是不可能立即编写出风格和结构具佳的程序的，一般的方法是采用自顶向下、逐步求精的模块化、结构化的方法进行分析和设计，把一个复杂问题变成若干便于实现的小问题。



# 典型问题1—多项序列求和

**例** 求序列：1, 3, 5, 7, 9.....的前  $n$  项之和。

```
#include<stdio.h>
int main()
{
 int n; // 项数
 int sum, term; //sum-和, term-某项

 sum = 0;
 term = 1; //默认为第一项
 scanf("%d", &n); // 输入项数n

 for (int i=1 ; i<=n; i++)
 {
 sum += term; // 累加一项
 term += 2; //计算下一项
 }
 printf(" sum = %d",sum);
 return 0;
}
```

**分析：**第 $i+1$ 项 = 第 $i$ 项 + 2

(1)定义项数 $n$ , 某项 $term$ , 求和变量 $sum$

(2) 设置初始值:  $sum=0$ ;  $term$ =第一项;

输入项数 $n$  (假设  $n=20$ ; )

(3)  $for(i=1; i<=n; i++)$  {

累加当前项; //  $sum += term$ ;

计算下一项; // $term += 2$ ;

}

(4) 输出 $sum$

- 分析上面的程序我们不难得出该程序的结构大致如下:

### 头文件部分

int main( )

{

    变量说明部分;

    初始化 ( 和清零, 项变量初始化为第一项)

    循环( 根据条件决定)

{

        累加一项;

        根据本项计算下一项;

}

    输出结果;

    return 0;

}

```
#include<stdio.h>
```

```
int main()
```

```
{
```

```
 int n; // 项数
```

```
 int sum, term; //sum-和, term-某项
```

```
 sum = 0;
```

```
 term = 1; //默认为第一项
```

```
 scanf("%d", &n); // 输入项数n
```

```
 for (int i=1 ; i<=n; i++)
```

```
 {
```

```
 sum += term; // 累加一项
```

```
 term += 2; //计算下一项
```

```
 }
```

```
 printf(" sum = %d",sum);
```

```
 return 0;
```

```
}
```

# 例2

求序列：1!, 2!, 3!, 4!.....的前n项之和。

第i+1项 = 第i项 \* (i+1)

头文件部分

```
int main()
```

```
{
```

变量说明部分;

初始化 ( 和清零, 项变量初始化第一项)

循环( 根据条件决定)

```
{
```

累加一项;

根据本项计算下一项;

```
}
```

输出结果;

```
return 0;
```

```
}
```

```
#include<stdio.h>
```

```
int main()
```

```
{
```

```
int n; //项数
```

```
long sum, term; //sum--和, term--某项
```

```
sum = 0;
```

```
term = 1; //默认为第一项
```

```
scanf("%d", &n); // 输入项数n
```

```
for (int i=1 ; i<=n ; i++)
```

```
{
```

```
sum += term;
```

```
term *= (i+1); // 计算下一项
```

```
}
```

```
printf(" sum = %ld",sum);
```

```
return 0;
```

```
}
```

# 比较例1和例2

求序列：1, 3, 5, 7, 9.....的前 n 项之和。

```
#include<stdio.h>
int main()
{
 int n; // 项数
 int sum, term; //sum--和, term--某项

 sum = 0;
 term = 1; //默认为第一项
 scanf("%d", &n); // 输入项数n

 for (int i=1 ; i<=n; i++)
 {
 sum += term; // 累加一项
 term += 2; //计算下一项
 }
 printf(" sum = %d",sum);
 return 0;
}
```

求序列：1!, 2!, 3!, 4!.....的前n项之和。

```
#include<stdio.h>
int main()
{
 int n; //项数
 long sum, term; //sum--和, term--某项

 sum = 0;
 term = 1; //默认为第一项
 scanf("%d", &n); // 输入项数n

 for (int i=1 ; i<=n ; i++)
 {
 sum += term;
 term *= (i+1); // 计算下一项
 }
 printf(" sum = %ld",sum);
 return 0;
}
```

(1) 变量说明不一样      (2) 计算当前项不一样

# 例3

求序列:  $\frac{1}{2}, \frac{3}{4}, \frac{5}{8}, \frac{7}{16}, \frac{9}{32}, \dots$ , 所有大于等于0.000001的数据项之和。

```
#include <stdio.h>
#include <math.h>
int main()
{
 int a, b; // 定义 a为分子, b为分母
 float sum = 0;

 a = 1; // 分子分母赋初值
 b = 2;

 while ((float)a/ b >= 1e-6)
 {
 sum += 1.0*a/ b; //累加一项
 a = a+2; //求下一项的分子
 b = b*2; //求下一项的分母
 }

 printf(" sum = %f",sum);
 return 0;
}
```

分析:

第i+1项分子 = 第i项分子+2

第i+1项分母 = 第i项分母\*2

运行结果

sum =2.999999

浮点数的舍入误差造成的现象

## 例4

```
#include <stdio.h>
#include <math.h>
int main()
{
 float x, sum, a, b ; //sum--和, a--分子, b--分母
 int s ; //符号控制变量
 printf("please input x:");
 scanf("%f", &x);
 a = x ; // 分子赋初值
 b = 1 ; // 分母赋初值
 s=1;
 sum = 0 ;
 for (int i=1; a/ b>=1e-6 ; i++)
 {
 sum += s* a/ b ; //累加一项
 a = a*x*x ; //求下一项的分子
 b = b*2*i*(2*i+1) ; //求下一项的分母
 s*=-1;
 }
 printf("sum = %f\n",sum);
 return 0;
}
```

计算  $\sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \frac{x^9}{9!} \dots$   
并使最后一项的绝对值小于 $1e-6$ 为止。

运行结果

please input x:2  
sum = 0.9092974

关键：分析答案需要满足什么条件，把满足条件的结果都输出来即可

► 例5

求100 - 999之间所有的水仙花数，所谓水仙花数就是说数的个十百位数的立方和恰好等于它本身。

```
#include<stdio.h>
int main()
{
 int a,b,c;

 for (int i=100;i<=999; i++)
 {
 a = i/100 ; //求百位数
 b =(i- a*100)/10; //求十位数 i/10%10
 c = i%10; //求个位数

 if(a*a*a + b*b*b +c*c*c == i)
 {
 printf(" %6d", i);
 }
 }
 return 0;
}
```

判断是否为水仙花数

运行结果

153 370 371 407

# 分析

程序的结构大致如下:

## 头文件部分

```
int main()
```

```
{
```

变量说明部分;

初始化 (可缺省);

循环( 根据条件决定)

```
{
```

预处理(可缺省);

根据条件判断输出所得结果  
(也可能是包含循环的程序结构);

```
}
```

```
return 0;
```

```
}
```

```
#include<stdio.h>
```

```
int main()
```

```
{
```

```
int a,b,c;
```

```
for (int i=100;i<=999; i++)
```

```
{
```

```
a = i/100 ; //求百位数
```

```
b =(i- a*100)/10; //求十位数
```

```
c = i%10; //求个位数
```

```
if(a*a*a + b*b*b +c*c*c == i)
```

```
{
```

```
printf(" %6d", i);
```

```
}
```

```
}
```

```
return 0;
```

```
}
```

这种循环条件加判断的编程方法称为**试数法**



**题目：** 小学生的加法算式

$$\begin{array}{r} a \ b \ c \\ + \ c \ b \ a \\ \hline 1 \ 3 \ 3 \ 3 \end{array}$$

a,b,c为一位数,试编程求所有可能a, b, c的值

a=4,b=1,c=9  
a=5,b=1,c=8  
a=6,b=1,c=7  
a=7,b=1,c=6  
a=8,b=1,c=5  
a=9,b=1,c=4

运行结果

```
#include<stdio.h>
int main()
{
 int a, b ,c;

 for (int i=100 ; i<=999; i++)
 {
 a = i/100 ; //求百位数
 b =(i- a*100)/10; //求十位数 i/10%10
 c = i%10; //求个位数

 if(1333 == a*100+b*10 + c + c*100 + b*10 + a)
 {
 printf("a= %d b=%d c=%d\n", a,b,c);
 }
 }
 return 0;
}
```

# 比较

求100 - 999之间所有的水仙花数。

```
#include<stdio.h>
int main()
{
 int a,b,c;

 for (int i=100;i<=999; i++)
 {
 a = i/100 ; //求百位数
 b =(i- a*100)/10; //求十位数
 c = i%10; //求个位数

 if(a*a*a + b*b*b +c*c*c == i)
 {
 printf(" %6d", i);
 }
 }
 return 0;
}
```

加法算式

```
#include<stdio.h>
int main()
{
 int a, b ,c;

 for (int j=100 ; j<=999; j++)
 {
 a = j/100 ; //求百位数
 b =(j- a*100)/10; //求十位数 j/10%10
 c = j%10; //求个位数

 if(1333 == a*100+b*10 + c + c*100 + b*10 + a)
 {
 printf("a= %d b=%d c=%d\n", a,b,c);
 }
 }
 return 0;
}
```

$$\begin{array}{r} a \ b \ c \\ + \ c \ b \ a \\ \hline 1 \ 3 \ 3 \ 3 \end{array}$$

仅仅是求解需要满足的条件不一样

# 例

求已知两个正整数的最大公约数。

please input a ,b:6 4  
the max =2

运行结果

```
#include<stdio.h>
int main()
{
 int a, b ;
 printf("please input a ,b:");
 scanf("%d%d", &a,&b);
 for (int i= a<b ? a:b ; i>0 ; i--)
 // i 初值为a,b中的较小值
 {
 if(a%i ==0 && b%i==0)
 {
 printf("the max = %d ", i);
 break;
 }
 }
 return 0;
}
```



# 枚举（穷举）法

穷举算法是程序设计中用得最为普遍、必须熟练掌握和正确运用的一种算法。它利用计算机运算速度快、精确度高的特点，**对问题的所有可能情况，一个不漏地进行检查，从中找出符合要求的答案。**

**用穷举算法解决问题，通常可从两方面进行分析：**

一、问题所涉及的情况：问题所涉及的情况有哪些，情况的种数可不可以确定。把它描述出来。

二、答案需要满足的条件：分析出来的这些情况，需要什么条件，才成为问题的答案。把这些条件描述出来。



# 枚举法举例

下列乘法算式中，每个汉字代表1个0~9的数字，不同的汉字代表不同的数字。试编程确定使得整个算式成立的数字组合，如有多种情况，请给出所有可能的答案。

$$\text{赛软件} * \text{比赛} = \text{软件比拼}$$



# 分析

赛软件 \* 比赛 = 软件比拼

a b c    d a    b c d e

这个算式中有5个不同的汉字，分别用变量a、b、c、d、e来代表，此算式可表示为：

$$(a*100+b*10+c) * (d*10+a) \\ == b*1000+c*100+d*10+e$$

a: 0~9    b: 0~9    c: 0~9    d: 0~9    e: 0~9

且取值不得重复，依次测试每一种取值情况，如果满足上式，则得到一个解，直到测试完毕，得到所有解。

```

for(a=0;a<=9;a++){ /* 枚举 a, 循环10次 */
 for(b=0;b<=9;b++){ /*枚举 b,循环9次*/
 if(b和a相等) continue;
 for (c=0; c<10; c++) { /*枚举 c,循环8*/
 if(c和a相等或者c和b相等) continue;
 for (d=0; d<10; d++) { /*枚举 d,循环7*/
 if(d和a~c中之一相等) continue;
 for (e=0;e<10; e++) { /*枚举e,循环6
 if(e和a~d中之一相等) continue;
 if (算式成立) 输出算式
 }
 }
 }
 }
}

```

```

for (a=0; a<10; a++) { /* 枚举 a */
 for (b=0; b<10; b++) { /* 枚举 b */
 if (b-a==0) /* b与a相等, 该b值跳过, 继续看下一个 b值 */
 continue;
 for (c=0; c<10; c++) { /* 枚举 c */
 if ((c-b)*(c-a) == 0) /* 等价于(c-b)==0 || (c-a)==0 */
 continue;
 for (d=0; d<10; d++) {
 if ((d-c)*(d-b)*(d-a) == 0)
 continue;
 for (e=0; e<10; e++) {
 if ((e-d)*(e-c)*(e-b)*(e-a) == 0)
 continue;
 if ((a*100+b*10+c) * (d*10+a) == b*100+c*100+d*10+e) /* 公式满足 */
 printf("%d%d%d * %d%d = %d%d%d%d\n", a, b, c, d, a, b, c, d, e);
 }
 }
 }
 }
}

```



# 例

**题目：**百钱百鸡问题，用100元钱买100只鸡，其中母鸡每只3元，公鸡每只2元，小鸡1元3只，且**每种鸡至少买一只**，试编一程序列出所有可能的购买方案。

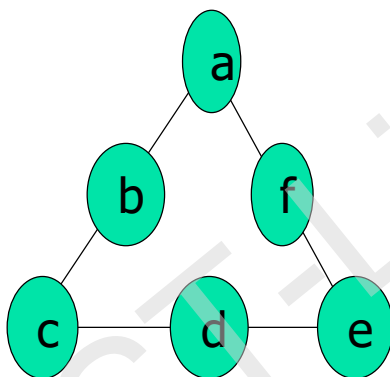
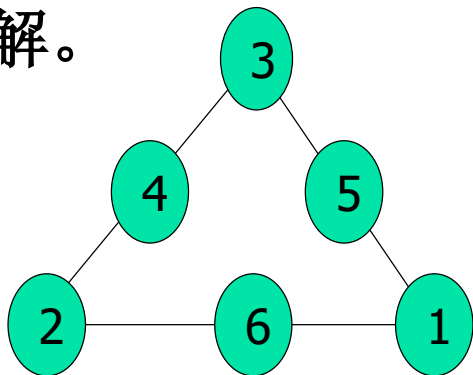
运行结果

母鸡数:5 公鸡数:32 小鸡数:63  
母鸡数:10 公鸡数:24 小鸡数:66  
母鸡数:15 公鸡数:16 小鸡数:69  
母鸡数:20 公鸡数:8 小鸡数:72

```
#include <stdio.h>
int main()
{
 int a, b, c; //a,b,c分别代表母鸡,公鸡和小鸡数
 for (a=1; a<=98;a++)
 {
 for (b=1;b<=98;b++)
 {
 for(c=1;c<=98;c++)
 {
 if ((a+b+c==100) &&(a*3+b*2+c/3==100)&&(c%3==0))
 printf("母鸡数:%d 公鸡数:%d 小鸡数:%d\n",a,b,c);
 }
 }
 }
 return 0;
}
```

# 枚举法练习

1、将1~6这六个整数排成如图所示的三角形，使得三角形三条边上的数字之和相等，求满足条件的的全部解。如图就是一个解。



穷举  $a, b, c, d, e$ ,  
算出  $f=21-(a+b+c+d+e)$

2、输入正整数 $x$  ( $2 \leq x \leq 79$ )，输出所有形如  $abcde/fghij=x$  的表达式，其中  $a \sim j$  由不同的0~9九数组成的。例如， $x=32$  时，输出为：

$$75168/02349=32$$

$$a/b=x$$

穷举  $b=1234 \sim 98765$ ，算出  $a=b*x$

if ( $a \leq 98765$  &&  $a$ 和 $b$ 没有相同的数字)

输出 算式

## 4.7

# 典型问题3—隐式条件问题

**例：** 编程实现一个最简单的计算器的功能，如输入3+5回车显示3+5=8；输错就退出（输入的不是加减乘除运算就判错）

**分析：** 根据输入运算符判断做何种操作

```
#include <stdio.h>
#include <math.h>
int main()
{
 float a, b, s; //定义两个操作数 及结果s
 char op; //定义运算符
 while (1)
 {
 printf("请输入两个数进行四则运算：\n");
 scanf("%f%c%f", &a, &op, &b);
 if ((op != '+') && (op != '-') && (op != '*') \
 && (op != '/'))
 break; //输错退出
 switch (op)
 {
 //根据输入的运算符进行相应的运算
 case '+':
 printf ("%g+%g=%g", a, b, a + b);
 break;
```

```
 case '-':
 printf ("%g-%g=%g", a, b, a - b);
 break;
 case '*':
 printf ("%g*%g=%g", a, b, a * b);
 break;
 case '/':
 if (fabs(b) < 1e-6) //确保除数不为零
 printf("除法错");
 printf ("%g/%g = %g", a, b, a / b);
 break;
 }
 }
 return 0;
}
```

# 例

**题目：** 编程序为小学生出一套最简单的整数(最大值小于100)加减乘运算的试题，共十道题，每道题随机产生，之后学生立即给答案，计算机立即判断正误，十道题做完给出成绩。

```
#include <stdio.h>
#include <time.h>
#include <stdlib.h>
```

```
#define N 10
```

```
int main()
{
 int a , b, s;
 char op ;
 int score=0;
 srand((unsigned) time(NULL));
 // 随机数发生器初始化
 for (int i =0; i<N; i++)
 {
 //随机产生一个0—100的整数
 a=rand()%100;
 b=rand()%100;
 // 随机产生一个0—2的整数, 0代表加,
 // 1代表减, 2代表乘
 op = rand()%3;
```

```
 if(op == 0) //0代表加
 {
 printf("\n%d + %d = ? ",a ,b);
 scanf("%d", &s);
 if (s==a+b)
 {
 printf("true");
 score =score+10;
 }
 else
 printf("false");
 } //end if
 else if(op == 1) // 1代表减
 {
 printf("\n%d - %d = ? ",a ,b);
 scanf("%d", &s);
 if (s==a-b)
 {
 printf("true");
 score =score+10;
 }
 else printf("false");
 } //end else if
```

```
 else if(op == 2) // 2代表乘
 {
 printf("\n%d * %d = ? ",a ,b);
 scanf("%d", &s);
 if (s==a*b)
 {
 printf("true");
 score =score+10;
 }
 else
 printf("false");
 } //end else if
 } //end for
 printf ("\nscore =%d ", score);
 return 0;
} //end main
```

**改为switch-case结构**

C语言产生随机数发生器函数: rand(); srand()

说明:

- 1、这两个函数都在头文件是stdlib.h中。
- 2、rand()函数和srand()函数必须配套使用。

用法举例:

原型: int rand(void); // 功能: 产生从0到RAND\_MAX 之间的随机整数。

头文件: stdlib.h

原型: void srand(unsigned seed) /\* 产生随机数的起始发生数据（种子），  
和rand函数配合使用\*/

头文件: stdlib.h time.h

# 伪随机数算法

- 伪随机数是指用数学**递推公式**所产生的随机数。
- 应用最广的**递推公式**：（即**线性同余法**）

$$a_0 = \text{seed}$$

$$a_n = (A * a_{n-1} + B) \% M, n \geq 1$$

其中 **A,B,M**是产生器设定的常数,用户不能更改。

**seed**: 种子参数, 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。

- ◆ 产生器给定了初始值
- ◆ **seed**应该在哪儿定义并初始化？
- ◆ 允许用户设置初始值（修改）
- ◆ 怎么修改？



# 库函数srand

- **srand**函数是随机数发生器的初始化函数，原型为：  
**void srand (unsigned int x) ;**   //原型在**stdlib.h**中  
// 用**x**作为种子，**rand**根据这个种子的值产生一系列随机数
- 用什么值做种子？  
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。  
**srand( time(NULL) );** //函数**time**的原型在**time.h**中

用法举例：产生10个0~99的随机整数

```
int main(void)
{
 int i;
 printf("Ten random numbers from 0 to 99\n\n");
 for (i=0; i<10; i++)
 printf("%d\n", rand()%100); //产生0~100的随机数
 return 0;
}
```

```
int main(void)
{
 int i;
 srand((unsigned) time(NULL)); //产生随机数的起始发生数据
 //time() 函数在头文件time.h中
 printf("Ten random numbers from 0 to 99\n\n");
 for (i=0; i<10; i++)
 printf("%d\n", rand()%100);
 return 0;
}
```

如何产生一个X到Y范围的随机数呢？

$$k = X + \text{rand()} \% (Y - X + 1);$$

每次运行程序产生的一组随机数都是一样的！因为随机数取在C语言中采用的是固定序列，所以每次执行所取的是同一个数。

运行程序，会发现每次产生的随机数都不一样！因为这里采用了时间作为种子，而时间在每时每刻都不相同，所以就产生了“随机”的随机数了。

要想产生不同的随机数，在使用rand之前需要先调用srand函数。



# 本章小结

- 注意**培养结构化的思维模式**，这对同学们今后的编程都大有好处。
- **学会归类一些典型问题**，掌握好典型问题的分析处理方法，套用现有的算法结构可以为我们分析新的问题减少许多工作量。
- 养成良好的编程习惯，主要有如下两点：
  - ① **关于程序风格**--培养良好的编程风格是学习程序设计的重要的一环。学会后要一如既往的坚持并保持风格的一致，程序的风格虽然都是些细小的事情，但对程序的质量有着至关重要的关系。
  - ② **关于结构化编程**--结构化编程是程序设计的基础，初学者要在掌握基本语法要点的基础上，**通过大量的编程实践**来熟练掌握条件语句和循环语句的用法。

# 判素数（自学）

判断整数 $m$ 是否素数。



凡不能被自身及1以外的整数整除的自然数

判一个数 $m$ 是否为素数的方法:

当用  $2, 3, \dots, (m-1)$  的整数去除 $m$ 时均不能除尽,  
则 $m$ 为素数。

$2, 3, \dots, (m/2)$

$2, 3, \dots, \sqrt{m}$



例如：判定9973是否素数。

最差的程序：要判断 $2 \sim 9972$ 不能整除

好一点：要判断 $2 \sim 4986$ 不能整除

最好的：用平方根，只要判断 $2 \sim 99$ 不能整除

一个数 $n$ 如果不是素数那么一定存在若干因子（不少于2个），

假设最小的因子是 $p$ ，

那么  $p * p \leq n$

所以  $p \leq \sqrt{n}$

即：一个比根号值大的数只可能和比根号值小的数同时成为因子，所以就只需要计算到比较小的那个数就够了

# 程序

```
#include<stdio.h>
```

```
int isprime(int m); /*函数原型 */
```

```
int main(void)
```

```
{ int m;
```

```
do{
```

```
 printf("input m:");
```

```
 scanf("%d" ,&m);
```

```
} while(m<2); // 输入一个大于等于2的整数
```

```
if(isprime(m)) /*函数调用 */
```

```
 printf("%d is a prime\n" ,m);
```

```
else
```

```
 printf("%d isn' t a prime\n" ,m);
```

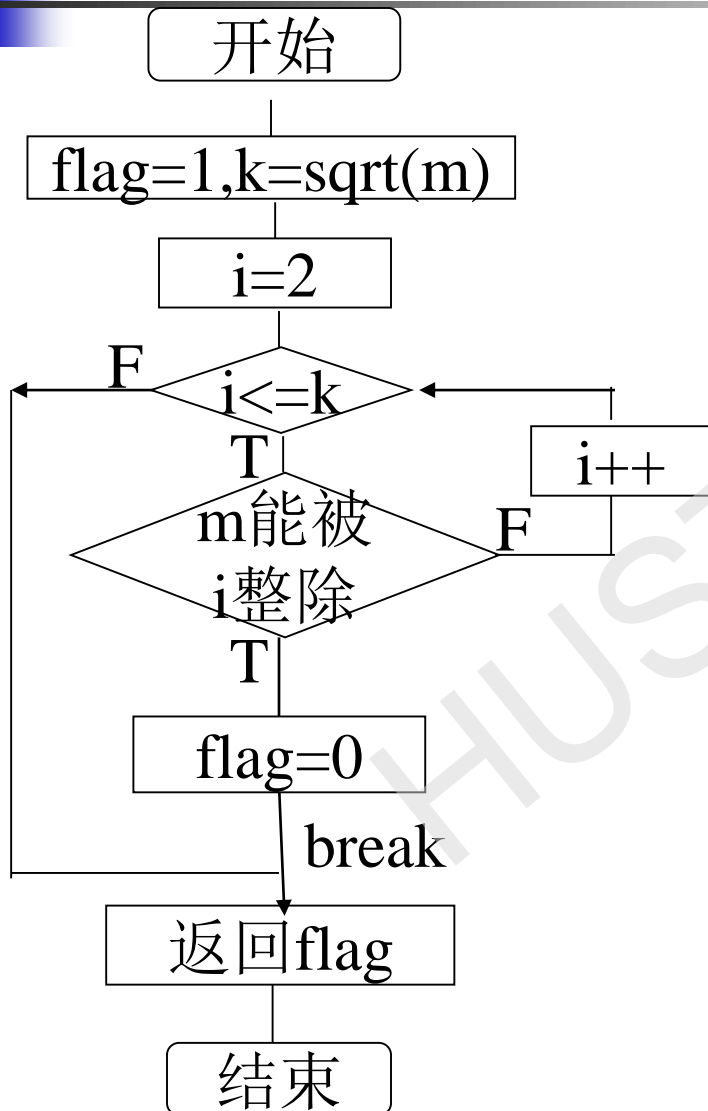
```
return 0;
```

```
}
```

将判断一个整数是否素数定义成函数isprime,是素数函数返回1, 否则, 返回0。

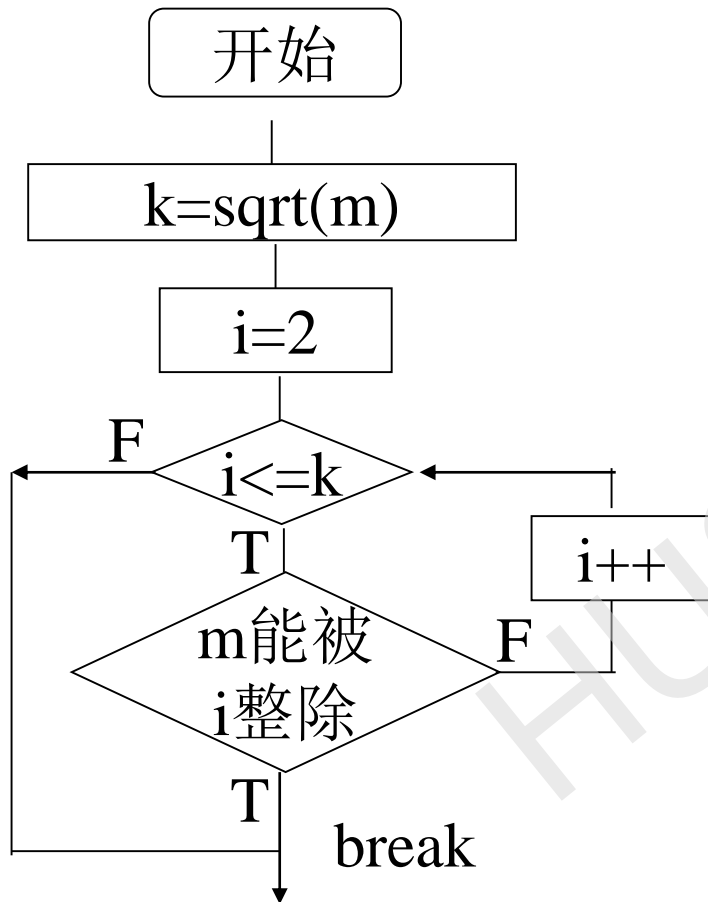
```
int isprime(int m);
```

# 函数 isprime



```
#include<math.h>
int isprime(int m)
{
 int flag,k,i;
 flag=1; // 初始是素数
 k=sqrt(m);
 for(i=2;i<=k;i++)
 if (!(m%i)) {
 flag=0;
 break;
 }
 return flag;
}
```

# 函数 isprime



```
#include<math.h>
int isprime(int m)
```

```
{
 int k,i;
 k=sqrt(m);
 for(i=2;i<=k;i++)
 if (!(m%i)) {
 break;
 }
 if(i>k) return 1;
 else return 0;
}
```

```
for (i=2;i<=k&&m%i;i++)
 ;
```



## 函数 isprime

```
#include<math.h>
int isprime(int m)
{
 int k,i;
 if(m==2) return 1;
 if(!(m%2)) return 0;
 k=sqrt(m);
 for(i=2;i<=k;i++)
 if (!(m% i)) return 0;
 return 1;
}
```

# 输出2~1000的所有素数

```
#include<stdio.h>
#define N 10 /* 每行输出的数据个数 */
int isprime(int m); /* 函数原型 */
int main()
{
 int n,k=1;
 printf("%5d",2); /* 2是素数，单独输出 */
 for (n=3; n<1000; n+=2) { /* 枚举每一个奇数 */
 if(isprime(n)) { /* 函数调用 */
 printf("%5d",n);
 if(!(++k/N)) putchar('\n'); /* 每输出N个数, 换行 */
 }
 }
 return 0;
}
```