

step by step

进阶

起步：

认知与体验（硬件、软件、程序与C语言）

进阶：

判断与推理（流程控制方法、语句）

抽象与联系（模块设计方法、函数）

表达与转换（基本操作、数据类型）

提高：

构造与访问（数组、指针、结构）

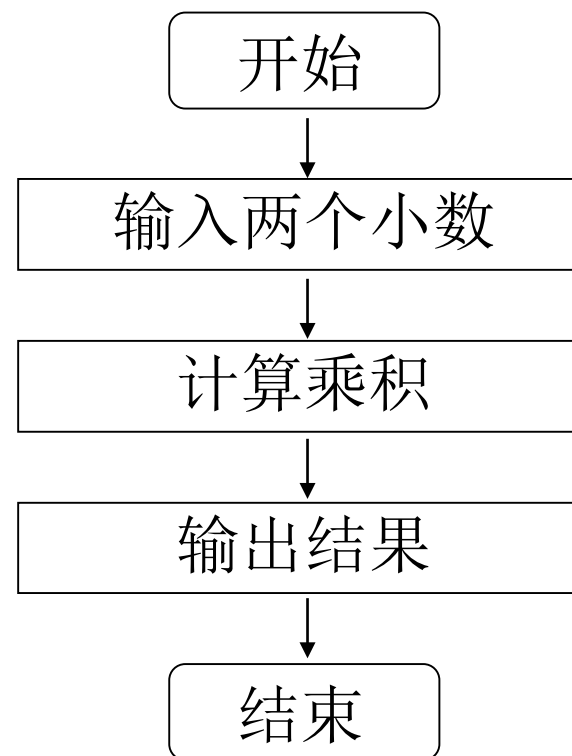
归纳与推广（程序设计的本质）

C语言的语句 (statement)

- 空语句：一个分号（最简单的语句，不执行任何操作）。
- 表达式语句：表达式末尾加一个分号。
 - `d = d + 1;`
- 关键字引导的简单语句：`while(表达式)`、`return`等带一个空语句或表达式语句。
 - `break;`
 - `return m*n;`
 - `if(x >= 0) y = 1; else y = 0;`
 - `while(i < 10) ++i;`
- 关键字引导的复合语句：`if(表达式){...}`、`switch(表达式){...}`、`do{...}while(表达式);`、`for(表达式;表达式;表达式){...}` 等带有花括号(将一个或多个语句括起来)的语句块。
 - ```
while(d < 10)
{
 sum = sum + PI * d;
 ++d;
}
```

## 程序的流程

- 顺序 (sequential flow)
- 分支 (conditional flow)
- 循环 (iteration flow, loop flow)



# 分支流程的基本形式及其控制语句

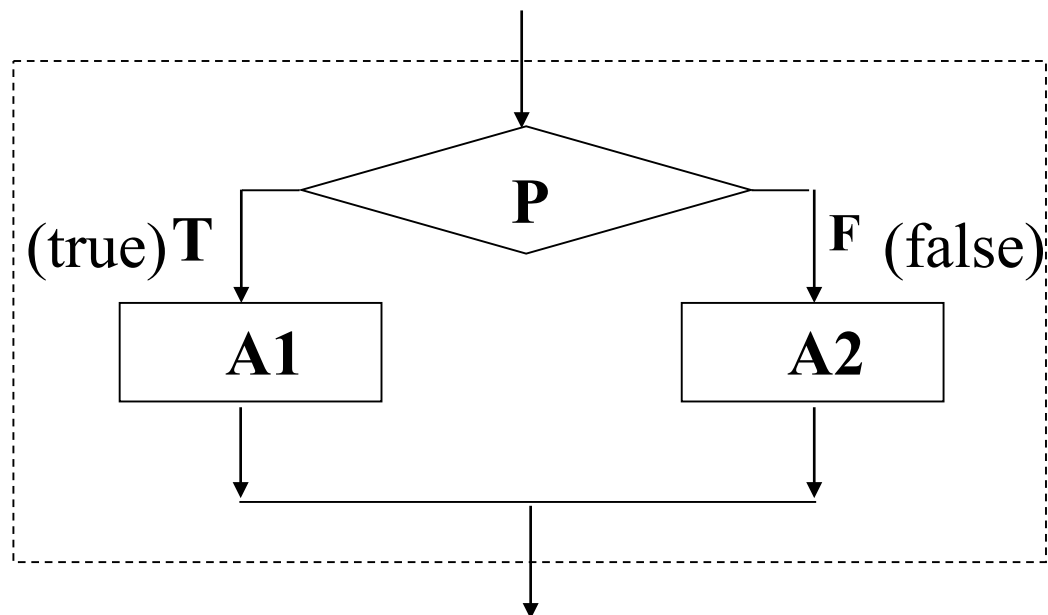
● 典型的分支流程，包含一个条件判断和两个分支任务。

➤ 先判断条件P

- 当条件P成立时，只执行任务A1，然后结束该流程；
- 当条件P不成立时，只执行任务A2，然后结束该流程。

```
if (<条件P>)
 <if子句>
else
 <else子句>
```

alternative clauses



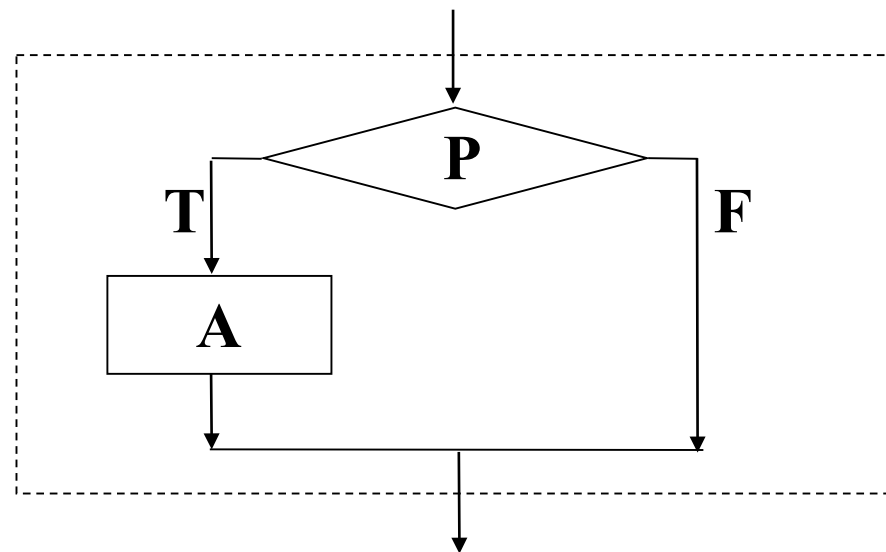
分支流程的另外一种形式，包含一个条件判断和一个分支任务。

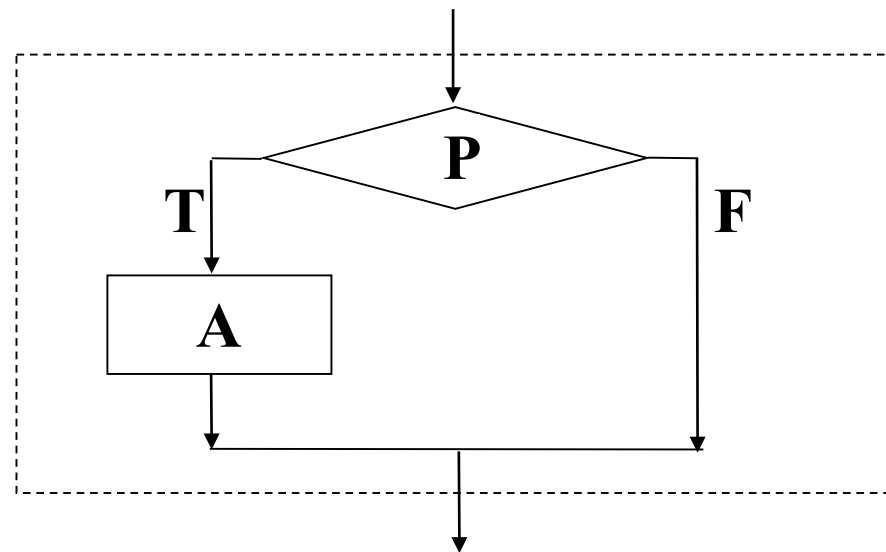
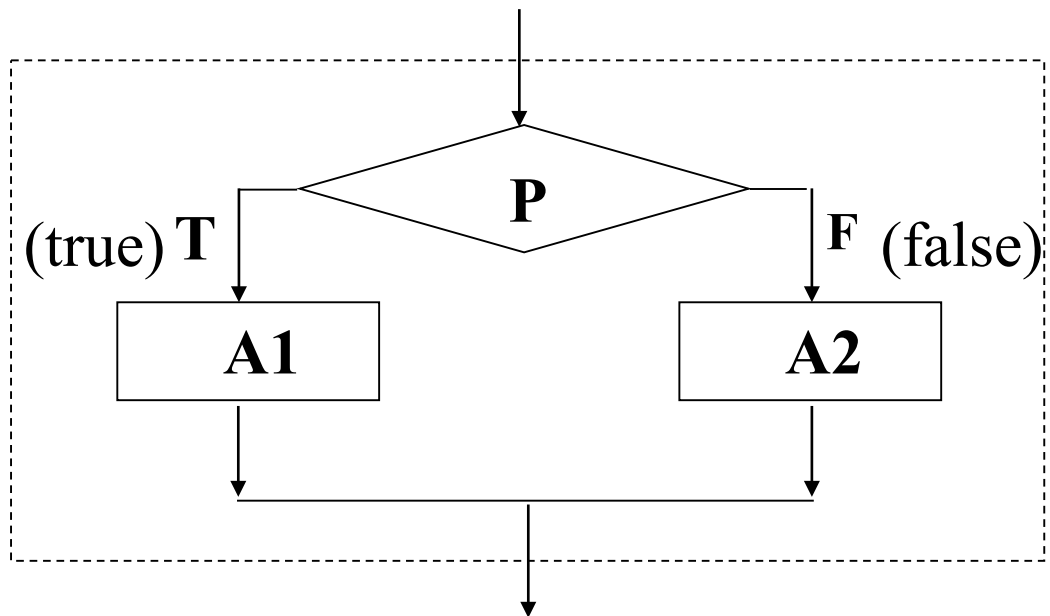
先判断条件P

- 当条件P成立时，执行任务A，然后结束该流程；
- 当条件P不成立时，不执行任务，然后结束该流程。

```
if (<条件P>)
 <if子句>
```

conditional clauses





分支流程中的条件只判断一次，每个任务最多只执行一次。

## 条件P

关系操作：比如 `if (n > 10)`

逻辑操作：比如 `if (n > 10 && n < 100)`

逻辑与操作，表示“而且”

---

● 例1.1 设计C程序，求输入的三个不相等的整数中的最大值，并输出。

● [分析问题]

- 该问题的求解需要分情况考虑，存在分支流程。先有一个分支流程判断前两个数哪个大，再有一个分支流程判断第三个数是不是还要大一些，从而得到最大值。

```
...
int main()
{
 int n1, n2, n3, max;
 scanf("%d%d%d", &n1, &n2, &n3);
 if(n1 > n2)
 max = n1;
 else
 max = n2;
 if(n3 > max)
 max = n3;
 printf("The max: %d \n" , max);
 return 0;
}
```

求出前两个数中的大数

判断谁是最大的数



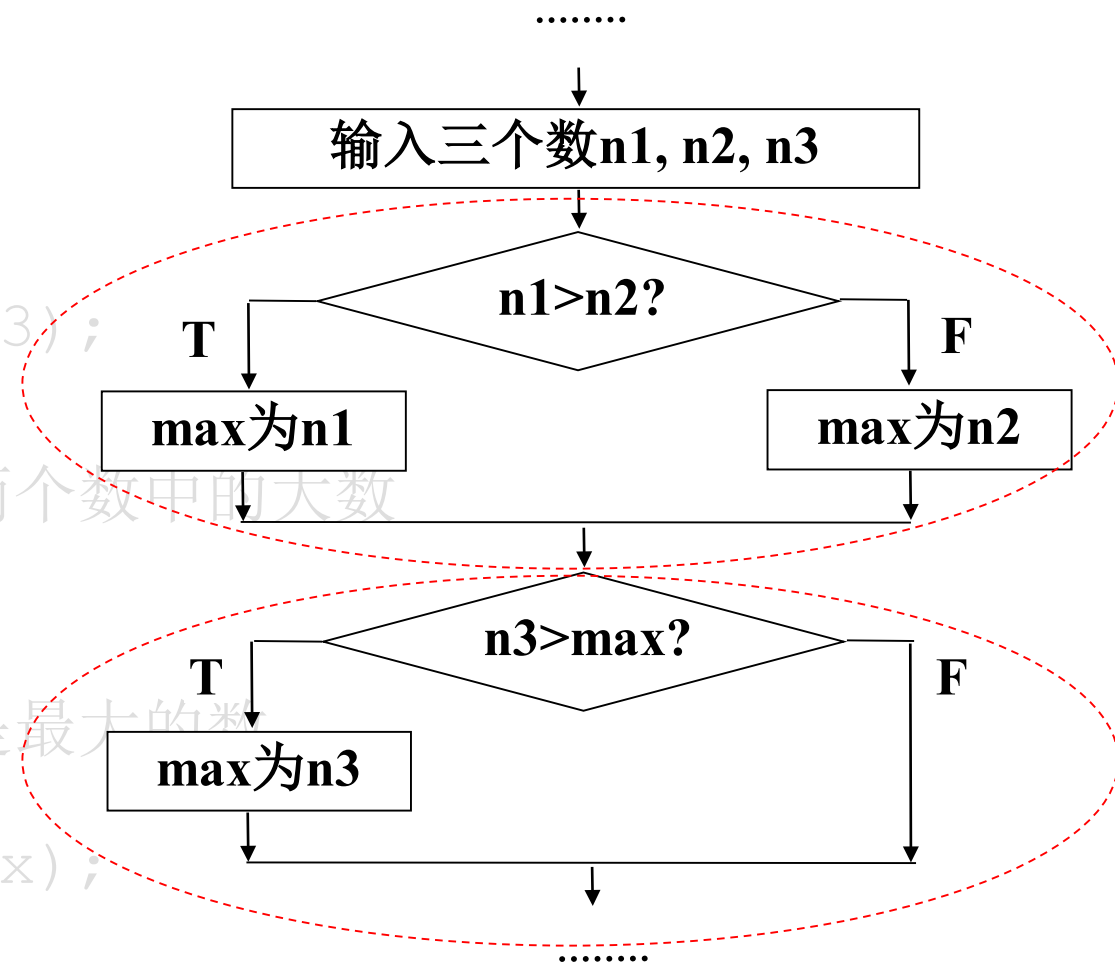
```

...
int main()
{
 int n1, n2, n3, max;
 scanf("%d%d%d", &n1, &n2, &n3);
 if(n1 > n2)
 max = n1;
 else
 max = n2;
 if(n3 > max)
 max = n3;
 printf("The max: %d \n" , max);
 return 0;
}

```

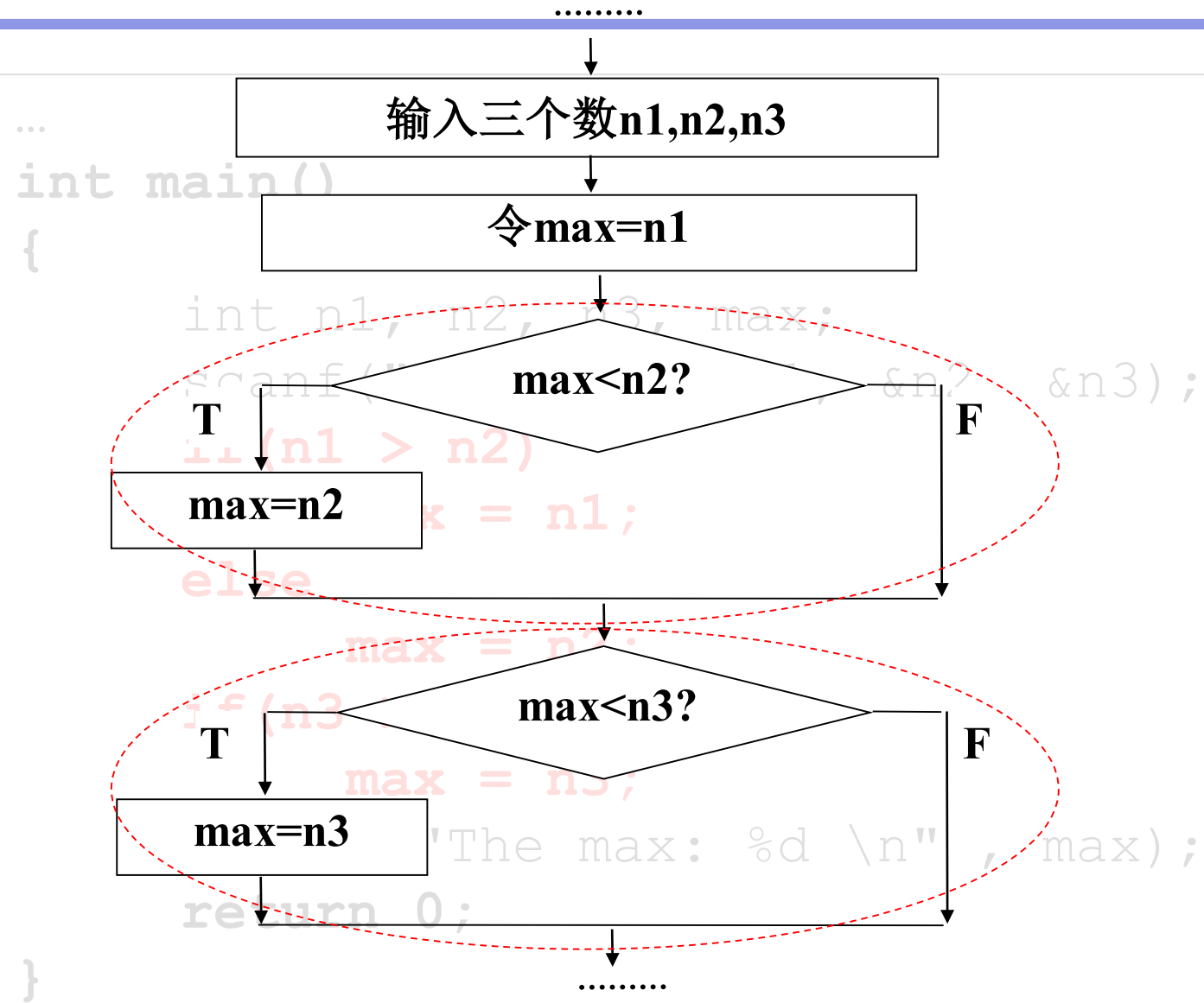
求出前两个数中的大数

判断谁是最大的数



```
...
int main()
{
 int n1, n2, n3, max;
 scanf("%d%d%d", &n1, &n2, &n3);
 if(n1 > n2)
 max = n1;
 else
 max = n2;
 if(n3 > max)
 max = n3;
 printf("The max: %d \n" , max);
 return 0;
}
```

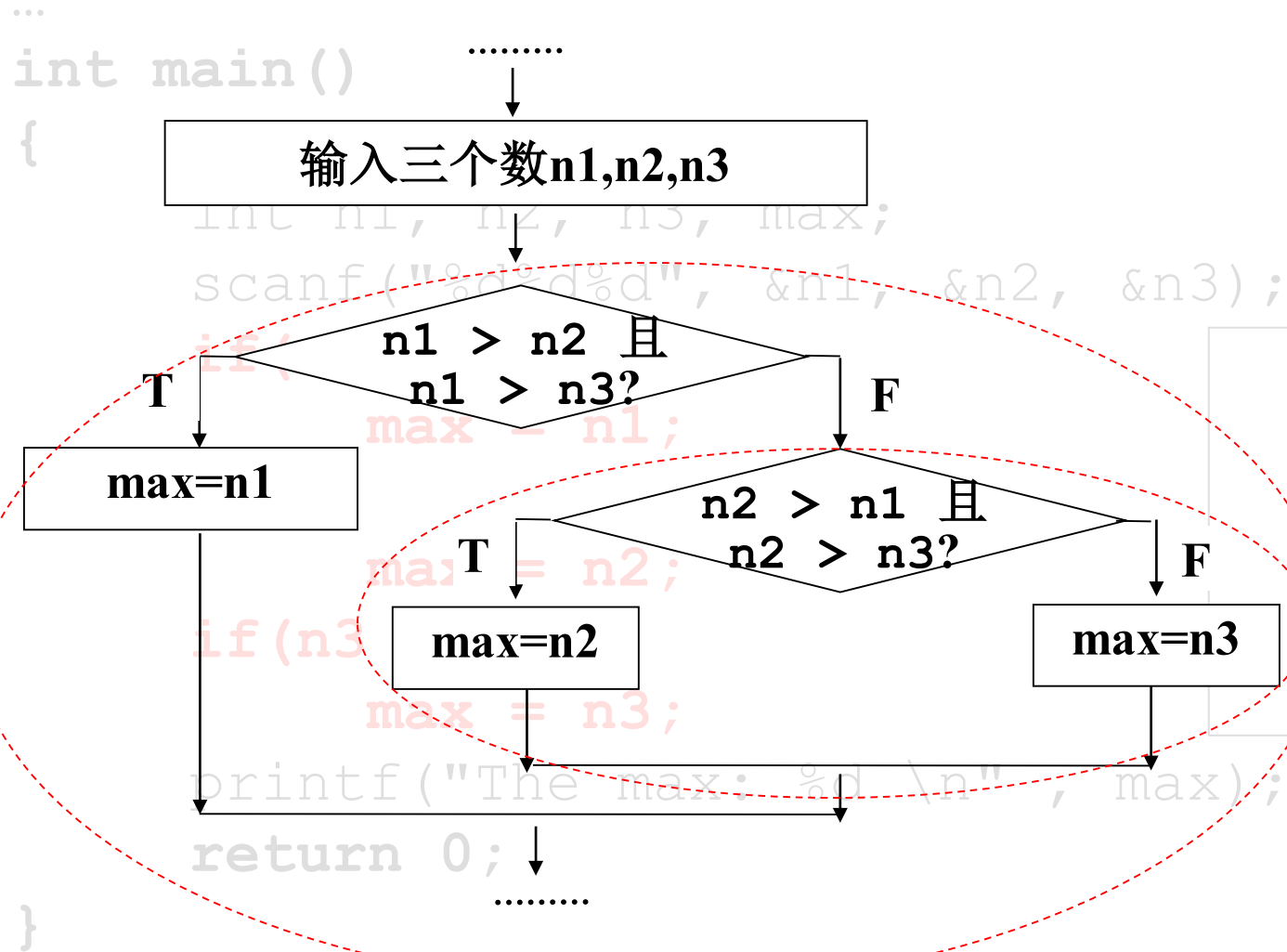
```
max = n1;
if(max < n2)
 max = n2;
if(max < n3)
 max = n3;
```



```
max = n1;
if (max < n2)
 max = n2;
if (max < n3)
 max = n3;
```

```
...
int main()
{
 int n1, n2, n3, max;
 scanf("%d%d%d", &n1, &n2, &n3);
 if(n1 > n2)
 max = n1;
 else
 max = n2;
 if(n3 > max)
 max = n3;
 printf("The max: %d \n" , max);
 return 0;
}
```

```
if(n1 > n2 && n1 > n3)
 max = n1;
else
 if(n2 > n1 && n2 > n3)
 max = n2;
 else
 max = n3;
```



```
if(n1 > n2 && n1 > n3)
 max = n1;
else
 if(n2 > n1 && n2 > n3)
 max = n2;
 else
 max = n3;
```

# 分支流程的书写

在一行的开头 按Tab键向右给出等量的空格，有的开发环境会自动帮程序员缩进。

## 编写if语句时，最好采用缩进形式。

- 好的缩进模式，且保持前后一致，不仅可以使程序美观，还有助于查看子句，提高程序的可读性。

```
if (x >= 0)
 y = x * x;
else
 printf("Input error! \n");
```

- 如果分支任务含多条语句，则一定要用一对花括号将它们组合成复合语句。

```
if (x >= 0)
{
 y = x * x;
 printf("%f * %f equal %f \n", x, x, y);
} //复合语句是一个整体，要么都被执行，要么都不被执行
else
 printf("Input error! \n");
```

- C程序中，当两种不同形式的if语句嵌套时，理解时会产生分歧。

```
max = n1;
if (n1 > n2)
 if (n3 > n1)
 max = n3;
else //这里else是对应 n3 > n1 不成立的情况
 // 不是对应 n1 > n2 不成立的情况

.....
```

- 缩进并不改变程序的逻辑。
- C语言规定，else子句与上面最近的、没有与else子句配对的if子句配对，而不是和较远那个if子句配对。

## 如果在逻辑上需要将else子句与较远的if子句配对

- 可以用一个花括号把较近的if子句写成复合语句

```
max = n1;
if (n1 > n2)
{
 if (n3 > n1)
 max = n3;
}
else /*这里else是对应n1 > n2不成立的情况*/

```

- 或者在较近的if子句后面用else和分号构造一个分支

```
max = n1;
if (n1 > n2)
 if (n3 > n1)
 max = n3;
 else /*这里else是对应n3 > n1不成立的情况*/
 ;
else /*这里else是对应n1 > n2不成立的情况*/

```



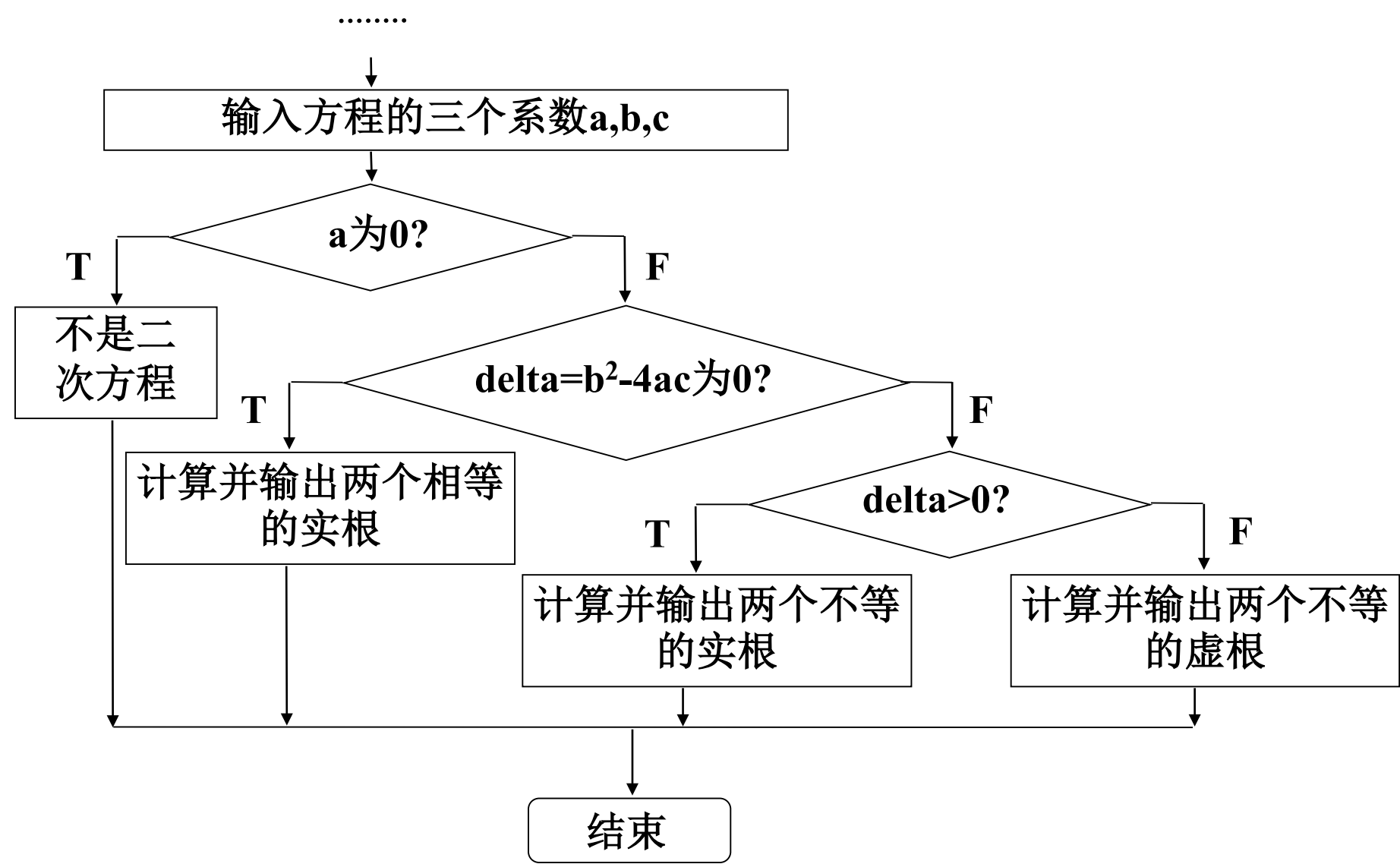
编辑嵌套的if语句时，更应采用结构清晰的缩进格式。不过，如果if语句嵌套层次很深，缩进会使代码过分偏右，给程序的编辑、查看带来不便。

```
if(score >= 90)
 printf("A \n");
else
 if(score >= 80)
 printf("B \n");
 else
 if(score >= 70)
 printf("C \n");
 else
 if(score >= 60)
 printf("D \n");
 else
 printf("Fail \n");
```

**建议改写成：**

```
if(score >= 90)
 printf("A \n");
else if(score >= 80)
 printf("B \n");
else if(score >= 70)
 printf("C \n");
else if(score >= 60)
 printf("D \n");
else
 printf("Fail \n");
```

● 例1.2 用求根公式求一元二次方程 $ax^2+bx+c=0$ 的根，并输出。



...

```
#include <math.h>
```

```
int main()
{
 double a, b, c, delta, p, q;
 printf("Please input three coefficients of \
the equation: \n"); //上一行续行符后不能有注释, 本行前没有空格
 scanf("%lf%lf%lf", &a, &b, &c);
 if(a == 0) // 这里切勿写成if(a = 0)
 printf("It isn't a quadratic equation! \n");
 else if((delta = b*b - 4*a*c) == 0)
 printf("x1 = x2 = %f \n", -b / (2 * a));
 else if(delta > 0)
 {
 p = -b / (2*a);
 q = sqrt(delta) / (2*a);
 printf("x1 = %f, x2 = %f \n", p + q, p - q);
 }
}
```

pow(x, y)

```
else
{
 p = -b / (2 * a);
 q = sqrt(-delta) / (2 * a);
 printf("x1 = %f + %fi, x2 = %f - %fi \n", p, q, p, q);
}
return 0;
}
```

在输出两个复数根时，采用先分别输出实部和虚部的方法，再添加+、-、i字符来表示复数。

Please input three coefficients of the equation:

1.2

2.1

3.4

x1 = -0.88+1.44i, x2 = -0.88-1.44i

# switch 语句 choice clauses

```
int week;
scanf("%d", &week);
switch (week) //该行没有分号
{
 case 0: printf("Sunday \n"); break;
 case 1: printf("Monday \n"); break;
 case 2: printf("Tuesday \n"); break;
 case 3: printf("Wednesday \n"); break;
 case 4: printf("Thursday \n"); break;
 case 5: printf("Friday \n"); break;
 case 6: printf("Saturday \n"); break;
 default: printf("error \n"); break;
} //该行没有分号
```

```
if (week == 0)
 printf("Sunday \n ");
else if (week == 1)
 printf("Monday \n ");
...
else
 printf("error \n ");
```

```
char grade;
grade = getchar();
switch (grade)
{
 case 'A': printf("90-100 \n"); break;
 case 'B': printf("80-89 \n"); break;
 case 'C': printf("70-79 \n"); break;
 case 'D': printf("60-69 \n"); break;
 case 'F': printf("0-59 \n"); break;
 default: printf("error \n"); break;
}
```

```
switch (week)
```

3

```
{
```

```
 case 1: printf("Monday \n");
```

```
 case 2: printf("Tuesday \n");
```

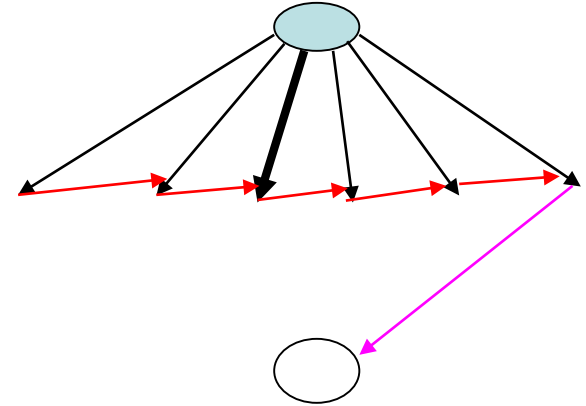
```
 case 3: printf("Wednesday \n");
```

```
 case 4: printf("Thursday \n");
```

```
 case 5: printf("Friday \n");
```

```
 default: printf("error \n");
```

```
}
```



Wednesday  
Thursday  
Friday  
error

开关

```
switch (week)
```

3

```
{
```

```
 case 1: printf("Monday \n"); break;
```

```
 case 2: printf("Tuesday \n"); break;
```

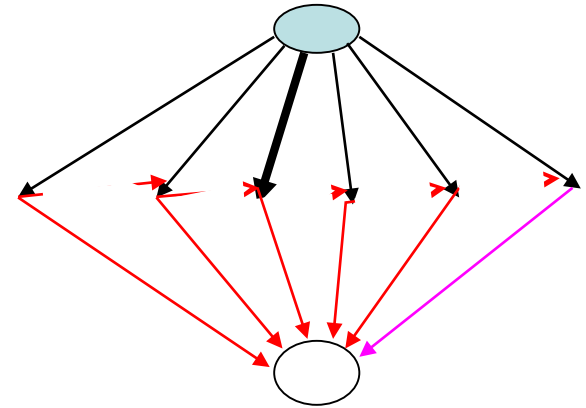
```
 case 3: printf("Wednesday \n"); break;
```

```
 case 4: printf("Thursday \n"); break;
```

```
 case 5: printf("Friday \n"); break;
```

```
 default: printf("error \n"); break;
```

```
}
```



Wednesday



- 某些语言（如：Pascal）的多分支语句中，一个分支执行完后将自动结束该流程。C语言的switch语句在一个分支执行完后，需要用break语句才能结束该流程，这样更具灵活性，当若干个分支具有部分重复功能时，可以节省代码量。

```
case 'A':
case 'B':
case 'C': printf(">60 \n "); break;
```

- 如果每个分支后面都有break语句，则分支可以按任意顺序排列，不过最好按易读的顺序排列。

```
switch(x)
{
 case 0: printf("xy = 0 \n "); break; // 外层分支
 case 1:
 switch(y)
 {
 case 0: printf("xy = 0 \n"); break; // 内层分支
 case 1: printf("xy = 1 \n"); break; // 内层分支
 default: printf("xy = %f \n", y); // 内层分支
 }
 break; // 外层分支
 default: printf("error! \n "); // 外层分支
}
```

# 多分支流程用 嵌套的 if-else 语句 还是 switch 语句?

---

## 🌈 判断条件

- 整数?
  - 字符?
  - 其他?
- } 试试 `switch`

---

分支流程 (conditional flow)



循环流程 (iteration/loop flow)

**起步:**

认知与体验 (硬件、软件、程序与C语言)

**进阶:**

判断与推理 (流程控制方法、语句)

抽象与联系 (模块设计方法、函数)

表达与转换 (基本操作、数据类型)

**提高:**

构造与访问 (数组、结构体、指针)

归纳与推广 (程序设计的本质)

# 循环流程的基本形式及其控制语句

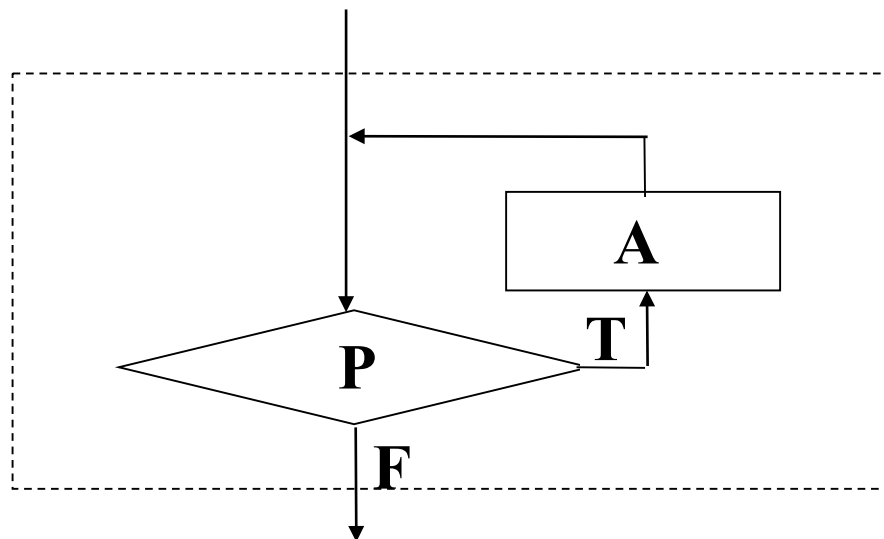
## 典型的循环流程包含一个条件判断和一个任务

### 先判断条件P

当型循环  
(while)

- 当条件P成立 (true) 时执行任务A (通常又叫循环体)，并再次判断条件P，如此循环往复；
- 当条件P不成立 (false) 时 (随着语句的执行，条件会从成立变为不成立)，该流程结束。

```
while (<条件P>)
 <任务A>
```



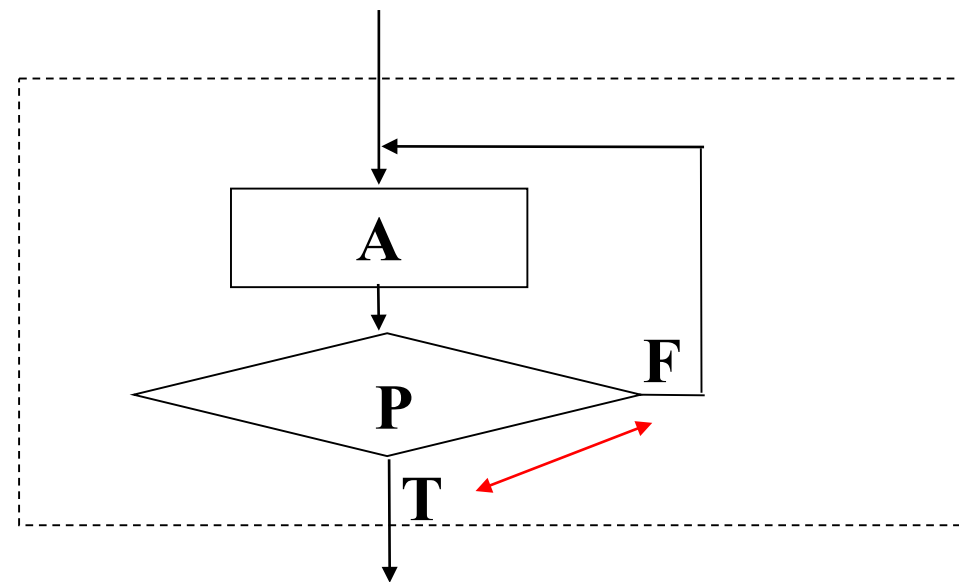
## ● 循环流程的另外一种形式，包含一个任务和一个条件判断

- 先执行一次任务A，再判断条件P
  - 当条件P不成立时继续执行任务A，并再次判断条件P，如此循环往复；
  - **直到**条件P成立时，该流程结束。

直到型循环  
(until)

C语言只能实现近似的直到型循环：

```
do
{
 <任务A>
}while(<条件P>) ;
```

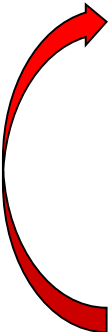


---

● 例1.3 设计C程序，求输入的一个整数以内所有自然数的和，并输出。

[分析] 求解这个问题的时候记不住求和公式没关系，可以用循环流程，依次把  $n$  个自然数累加到存储和的变量中。

```
...
int main()
{
 int n;
 scanf("%d", &n);
 int i = 1, sum = 0;
 while(i <= n) //该行没有分号
 {
 sum = sum + i; //可改为“sum += i;”
 i = i + 1; //可改为“++i;”
 }
 printf("Sum. of integers 1-%d: %d\n", n, sum);
 return 0;
}
```

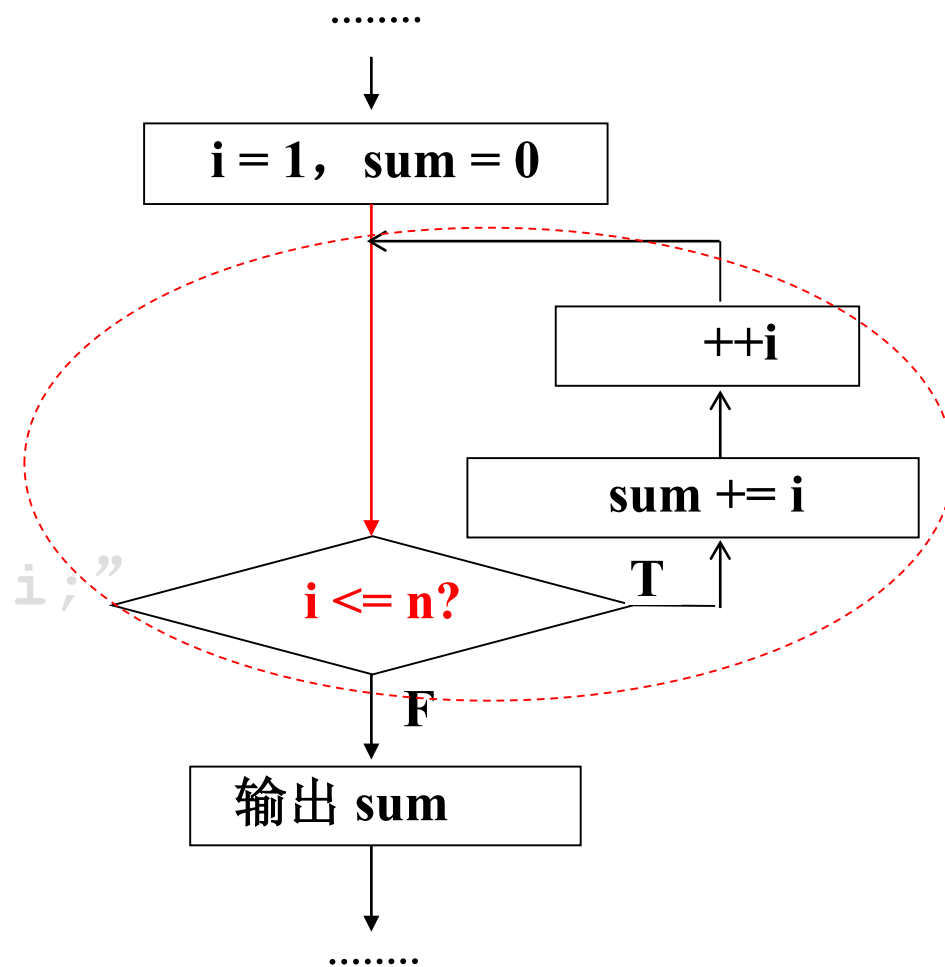




```

...
int main()
{
 int n;
 scanf("%d", &n);
 int i = 1, sum = 0;
 while(i <= n) //该行没有分号
 {
 sum = sum + i; //可改为“sum += i;”
 i = i + 1; //可改为“++i;”
 }
 printf("Sum. of integers 1-%d: %d\n", n, sum);
 return 0;
}

```



---

```
...
int main()
{
 int n;
 scanf("%d", &n);
 int i = 1, sum = 0;
 do
 {
 sum += i;
 ++i;
 }while(i <= n); //该行有分号

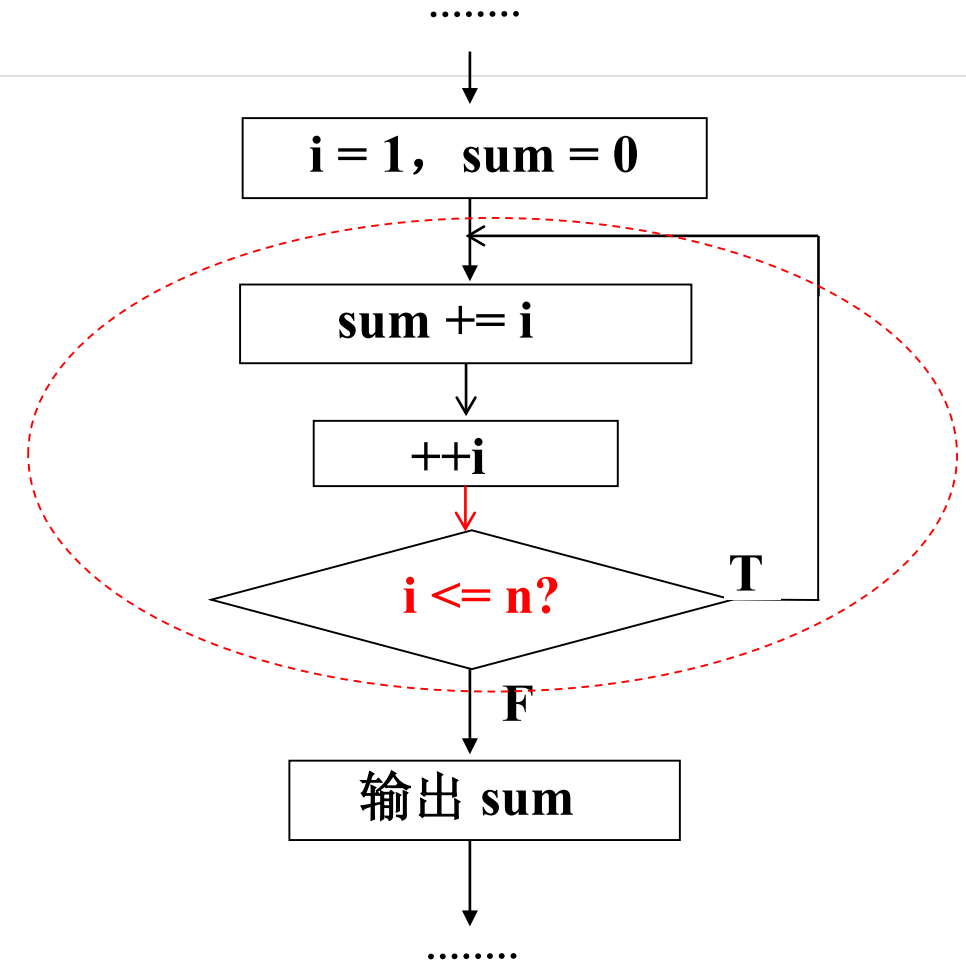
 printf("Sum. of integers 1-%d: %d\n", n, sum);
 return 0;
}
```

```

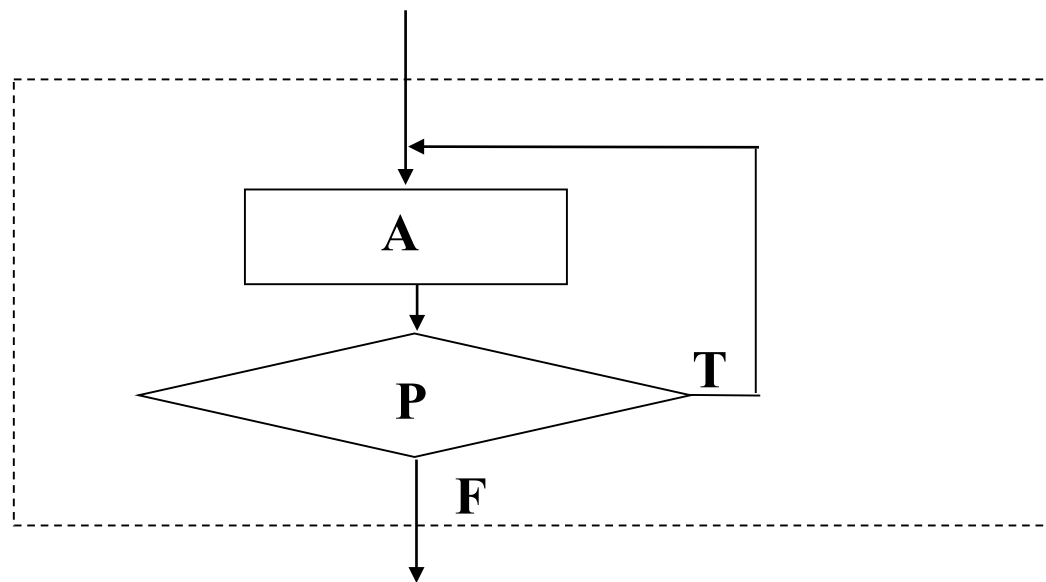
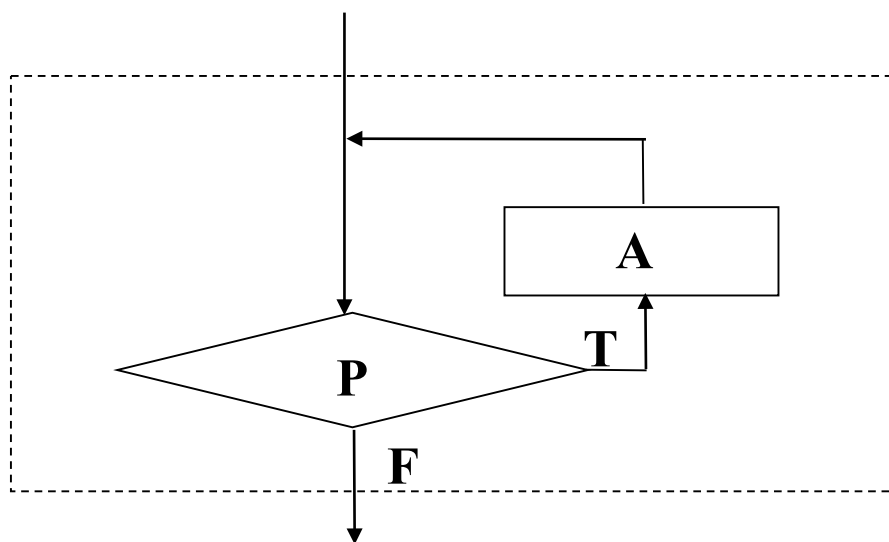
...
int main()
{
 int n;
 scanf("%d", &n);
 int i = 1, sum = 0;
 do
 {
 sum += i;
 ++i;
 }while(i <= n); //该行有分号

 printf("Sum. of integers 1-%d: %d\n", n, sum);
 return 0;
}

```



# 两类循环的异同点



- 条件P至少判断一次，并在执行任务A之后继续判断下一次；任务A可能执行有限次（条件P存在不成立的可能），也可能执行无限次，即死循环（条件P一直成立）。**所以要注意条件P的设计，避免循环不能正确执行或死循环。**
- 条件P一开始不成立的情况下，先判断的循环一次任务也不执行，而后判断的循环会执行一次任务。

=101 ?

```
int i=1, sum=0;
while(i <= 100)
{
 sum += i;
 ++i;
}
```

```
int i=1, sum=0;
while(i <= 100)
{
 sum += i;
} //?
```

=101 ?

```
int i=1, sum=0;
do
{
 sum += i;
 ++i;
}
while(i <= 100);
```

```
int i=1, sum=0;
do
{
 sum += i;
}
while(i <= 100); //?
```

# while语句和do...while语句的书写

## 多写或少写分号

```
while (i <= N); //死循环
{
 sum += i; //该行不属于循环体
 ++i; //该行不属于循环体
}
```

```
do
{
 sum += i;
 ++i;
}while (i <= 100)
//语法错误
```

如果条件成立时要执行多个语句，则一定要用花括号把这些语句写成复合语句的形式，否则，编译错/或结果不正确/甚至出现死循环，因为缩进并不改变程序的逻辑。

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

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

# for 语句

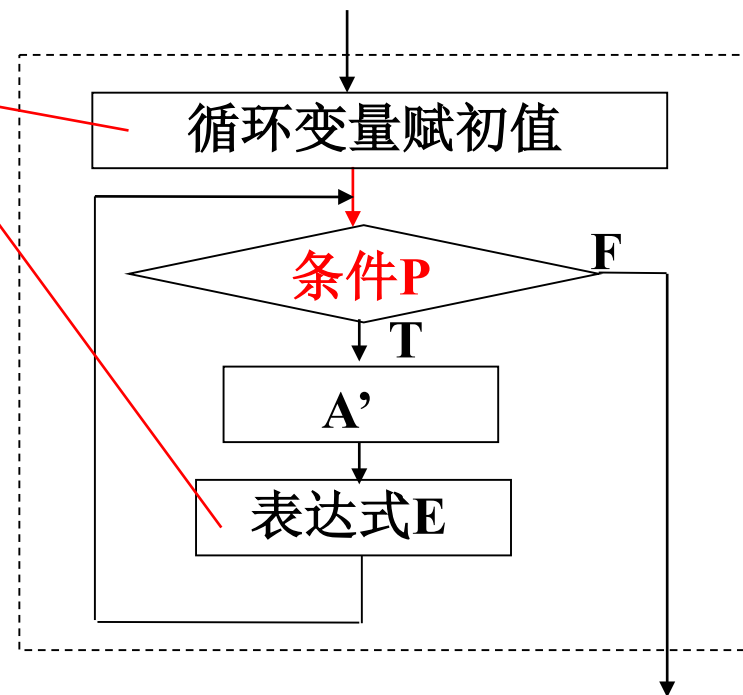
for(<循环变量赋初值>;<条件P>;<表达式E>)

{

<任务A' >

}

- 先对循环变量赋初值，再判断条件P
  - 当条件P成立时，执行任务A'，并计算表达式E，然后再判断条件，如此循环往复；
  - 当条件P不成立时，结束该流程。

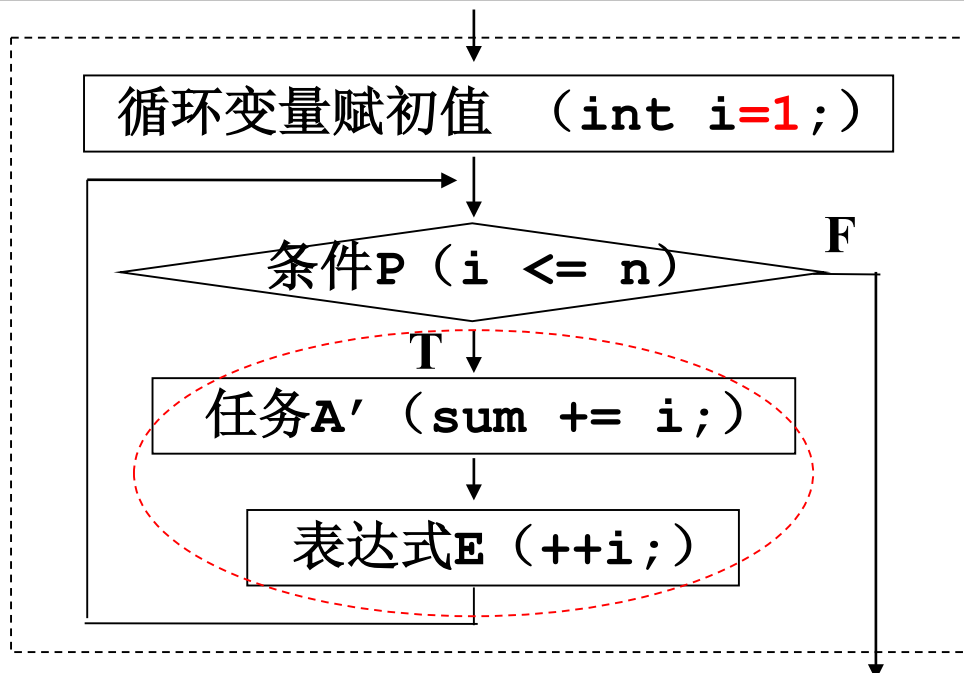


# for 语句

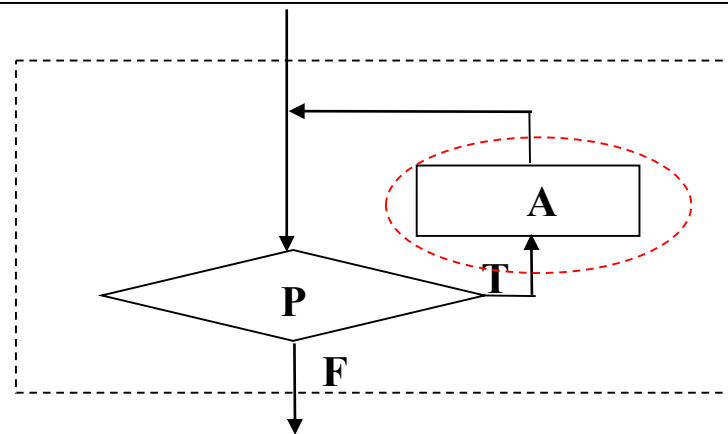
```
int sum=0;
for(int i=1; i <= n; ++i)
 sum += i;
```

$i += 2$

循环体只有一条语句，花括号可不加



```
int i=1, sum=0;
while(i <= n)
{
 sum += i;
 ++i;
}
```



for语句一般将循环变量放入for内赋初值（while或do...while语句的循环变量通常在while或do前赋初值）；表达式E一般是修改循环变量的操作，可以明显看出步长，循环变量按步长增大或减小，促使循环结束；任务A'和E合起来相当于while或do...while语句中的任务A。



# 控制循环流程用while、do-while还是for语句？

- 从表达能力上讲，上述三种语句是等价的，可以互相替代。

- 一般原则：

- 计数控制的循环（counter-controlled loop），用for语句
- 事件控制的循环（event-controlled loop），一般使用while或do-while语句
- 如果循环体至少执行一次，则使用do-while语句。

|                  |
|------------------|
| <b>bounded</b>   |
| <b>unbounded</b> |

# 利用循环提高程序的鲁棒性

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

```
int main()
```

```
{
```

```
 double r;
```

输入负数?

```
 scanf("%lf", &r);
```

```
 double s = 3.14*r*r;
```

```

```

```
 return 0;
```

```
}
```

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

```
int main()
```

```
{
```

```
 double r;
```

```
 scanf("%lf", &r);
```

```
 while(r <= 0)
```

```
 {
```

```
 scanf("%lf", &r);
```

```
 }
```

```
 double s = 3.14*r*r;
```

```

```

```
 return 0;
```

```
}
```

事件控制的循环

# 防止输入错导致的死循环\*

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

```
int main()
{
```

```
 double r = 0;
 scanf("%lf", &r);
 while(r <= 0)
 {
 scanf("%lf", &r);
 }
 double s = 3.14*r*r;

 return 0;
}
```

对于非字符型变量 `r`，程序运行时若输入字符，`r` 获取不到输入数据，`scanf` 库函数将返回 0（正常情况返回 1），导致之后的 `scanf` 库函数不再被执行，从而有可能导致死循环。

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

```
 double r = 0;
 if(! scanf("%lf", &r))
 return -1;
 while(r <= 0)
 {
 if(! scanf("%lf", &r))
 return -1;
 }
 double s = 3.14*r*r;

 return 0;
```

为了防止这种情况，可添加判断和提前终止程序语句。

# 循环流程的嵌套

- 循环流程也可以嵌套，即循环体中又含有循环流程。

```
for(int i = 1; i <= 9; ++i)
{
 for(int j = 1; j <= 9; ++j)
 printf("%d \t", j);
 printf("\n");
}
```

制表符

```
#include <iomanip>
cout << setw(8) << i*j;
```

|   |   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

## 例1.4 输出一个九九乘法表。

...

```
int main()
{
 printf(" Multiplication Table \n");
 for(int i = 1; i <= 9; ++i)
 {
 for(int j = 1; j <= 9; ++j)
 printf("%d \t", i * j);
 printf("\n");
 }
 return 0;
}
```

| Multiplication Table |    |    |    |    |    |    |    |    |  |
|----------------------|----|----|----|----|----|----|----|----|--|
| 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 |  |

嵌套关系

```
for(int i = 1; i <= 9; ++i)
{
 for(int j = 1; j <= 9; ++j)
 {
 if(j < i)
 printf(" \t");
 else
 printf("%d \t", i * j);
 }
 printf("\n");
}
```

并列关系

```
.....
for(int i = 1; i <= 9; ++i)
{
 for(int j = 1; j < i; ++j)
 printf(" \t");
 for(int j = i; j <= 9; ++j)
 printf("%d \t", i * j);
 printf("\n");
}
```

```
return 0;
```

Multiplication Table

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

## 例1.5 求输入的一个正整数的阶乘并输出。

---

...

```
int main()
{
 int n, i = 2, f = 1; //f 要初始化!

 scanf("%d", &n);
 while(i <= n)
 {
 f *= i; //相当于f = f * i;
 ++i;
 }
 printf("%d \n", f);
 return 0;
}
```

## 例1.5' 每输入一个正整数，输出其阶乘，直到输入0。

```
int main()
{
 int n, i, f;
 scanf("%d", &n);
 while(n != 0) // != 表示“不等于”
 {
 i = 2, f = 1;
 while(i <= n)
 {
 f *= i;
 ++i;
 }
 printf("%d \n", f);

 scanf("%d", &n);
 }
 return 0;
}
```

外循环是事件控制型循环，即其循环条件是“n != 0”这个事件是否发生 (while/do-while更适合)；

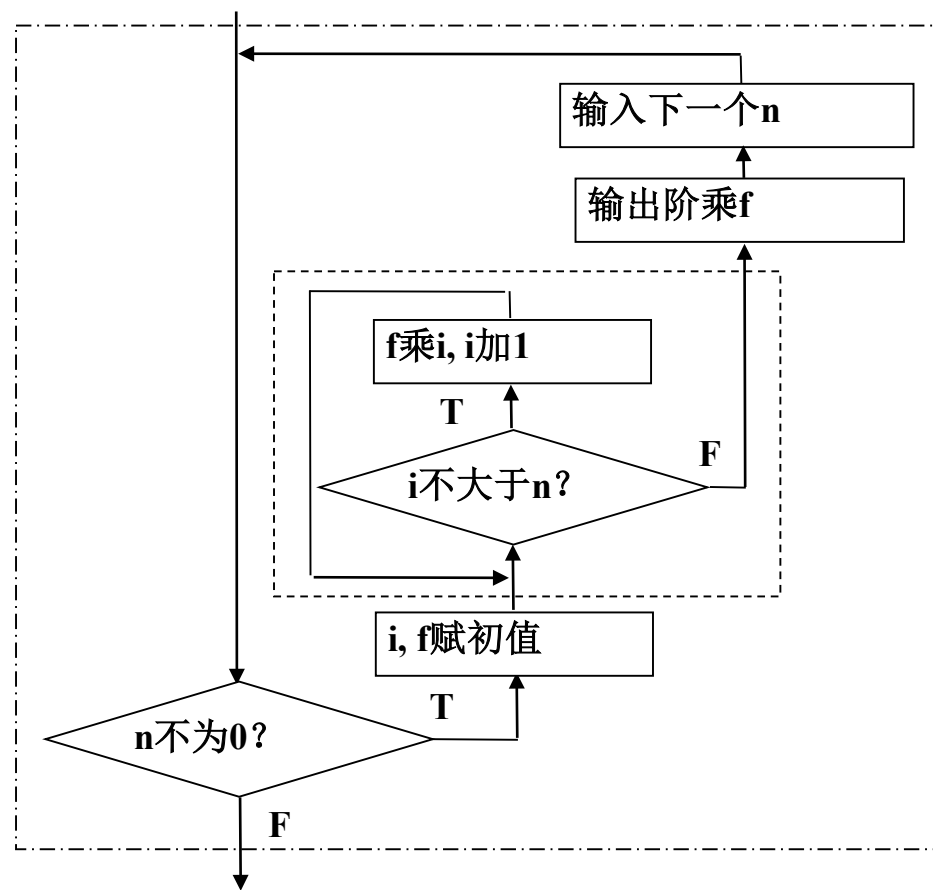
内循环是计数控制型循环，其循环条件是计数变量 i 是否达到边界值 n (for更适合)。



## 例1.5' 每输入一个正整数，输出其阶乘，直到输入0。

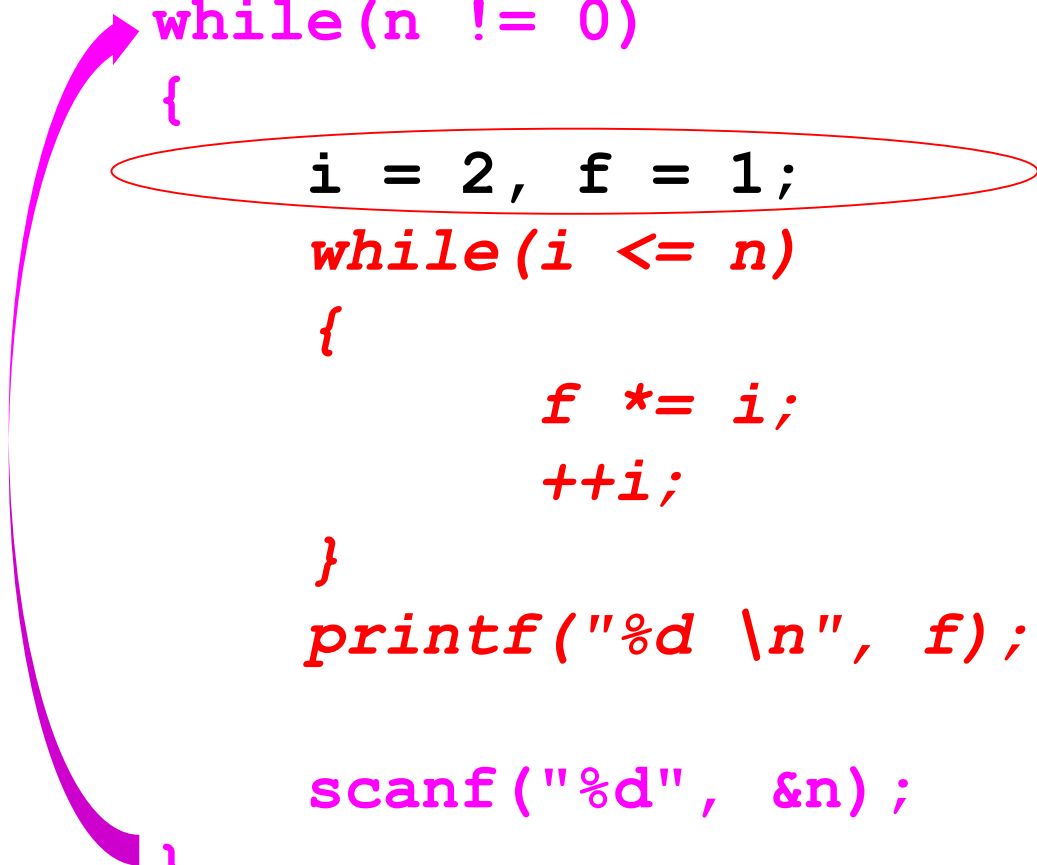
```
int main()
{
 int n, i, f;
 scanf("%d", &n);
 while(n != 0)
 {
 i = 2, f = 1;
 while(i <= n)
 {
 f *= i;
 ++i;
 }
 printf("%d \n", f);

 scanf("%d", &n);
 }
 return 0;
}
```



```
int main()
{
 int n, i = 2, f = 1;
 scanf("%d", &n);
 while(n != 0)
 {
 i = 2, f = 1;
 while(i <= n)
 {
 f *= i;
 ++i;
 }
 printf("%d \n", f);

 scanf("%d", &n);
 }
 return 0;
}
```



# 循环的优化

## 编译器自动优化

```
for(int i=0; i < 10000; ++i)
 s = a+b;
```

```
int sum = 0;
```

```
for(int i=1; i <= 100; ++i)
 ++sum;
```

空语句

## 如果需要实现延时，就关闭编译器的优化功能(项目-属性-优化-Disabled)

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

➤ 如果要延长比较长的时间，可以用嵌套的循环

## 程序员也可以有意识地优化循环程序，以便提高程序的运行效率。

### 提取与循环无关的计算

```
int s=0, m, n;
scanf("%d%d", &m, &n);
for(int i = 0; i < m*n-i; ++i)
 s += i;
```

### 可以优化成：

```
int s=0, m, n;
scanf("%d%d", &m, &n);
int l = m*n;
for(int i = 0; i < l-i; ++i)
 s += i;
```

## ◆ 嵌套循环应遵循“外小内大”原则

```
for(int i = 0; i < 1000; ++i)
 for(int j = 0; j < 10; ++j)
 s += i*j;
```

可以优化成:

```
for(int j = 0; j < 10; ++j)
 for(int i = 0; i < 1000; ++i)
 s += i*j;
```

- ◆ 如果循环次数很大，尽量不在循环流程里嵌套分支流程

```
for (i = 0; i < N; ++i)
{
 if (...)
 A1;
 else
 A2;
}
```

如果能保证功能等价，就改写成分支流程嵌套循环流程的形式：

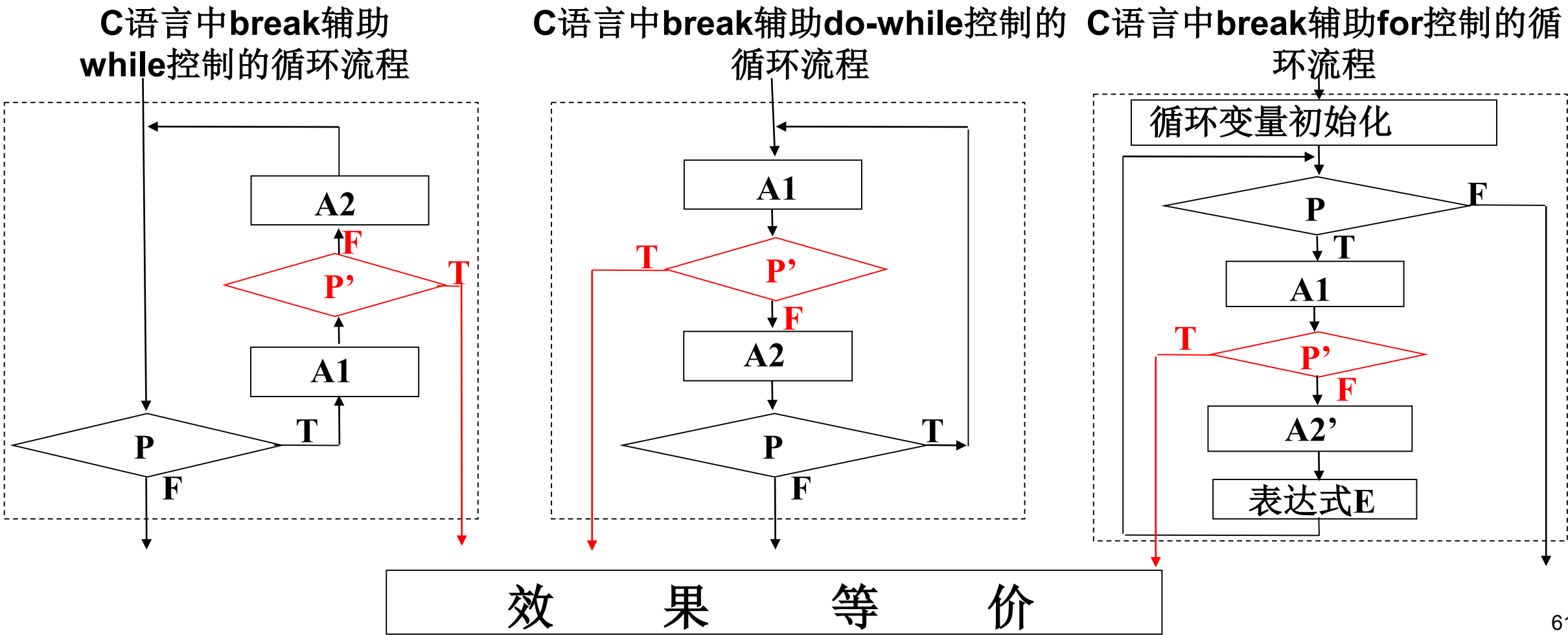
```
if (...)
 for (i = 0; i < N; ++i)
 A1;
else
 for (i = 0; i < N; ++i)
 A2;
```

- ◆ 如果循环次数不大，改写后效率提高不明显，不必改写，以保持程序简洁。

- 
- ◆ 如果循环体中处理的数据量较大，应结合数据的**存储**情况综合考虑循环流程的优化
    - 一次计算涉及的操作数不能同时进入缓存会严重降低计算效率
    - 《计算机系统基础》课程内容

# 循环流程的折断(break)

● 即循环操作往往被分成两部分，然后根据一定情况在相应的语句控制下，在执行其中一部分操作后，结束整个循环，从而提高循环流程的灵活性。

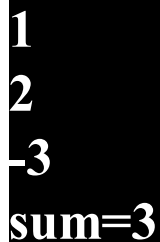




# break

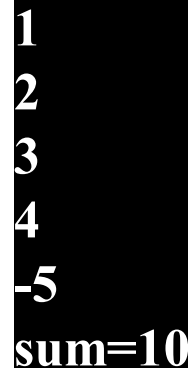
- 例1.6 设计C程序，求输入的10个整数的和，遇到负数或0就提前终止。

```
int d, sum = 0, i = 1;
while(i <= 10)
{
 scanf("%d", &d);
 if(d <= 0) break; //结束循环流程
 sum += d;
 ++i;
} ▲
printf("sum: %d \n", sum);
```



```
1
2
-3
sum=3
```

这次运行，  
循环只完整  
地执行了2次。



```
1
2
3
4
-5
sum=10
```

这次运行，  
循环只完整  
地执行了4次。

## ● 例1.6 设计C程序，求输入的10个整数的和，遇到负数或0就提前终止。

```
int d, sum = 0, i = 1;
while(i <= 10)
{
 scanf("%d", &d);
 if(d <= 0) { ; break; } //结束循环流程
 sum += d;
 ++i;
} ▲
printf("sum: %d \n", sum);
```

```
1
2
-3
sum=3
```

这次运行，  
循环只完整地  
执行了2次。

```
1
2
3
4
-5
sum=10
```

这次运行，  
循环只完整地  
执行了4次。

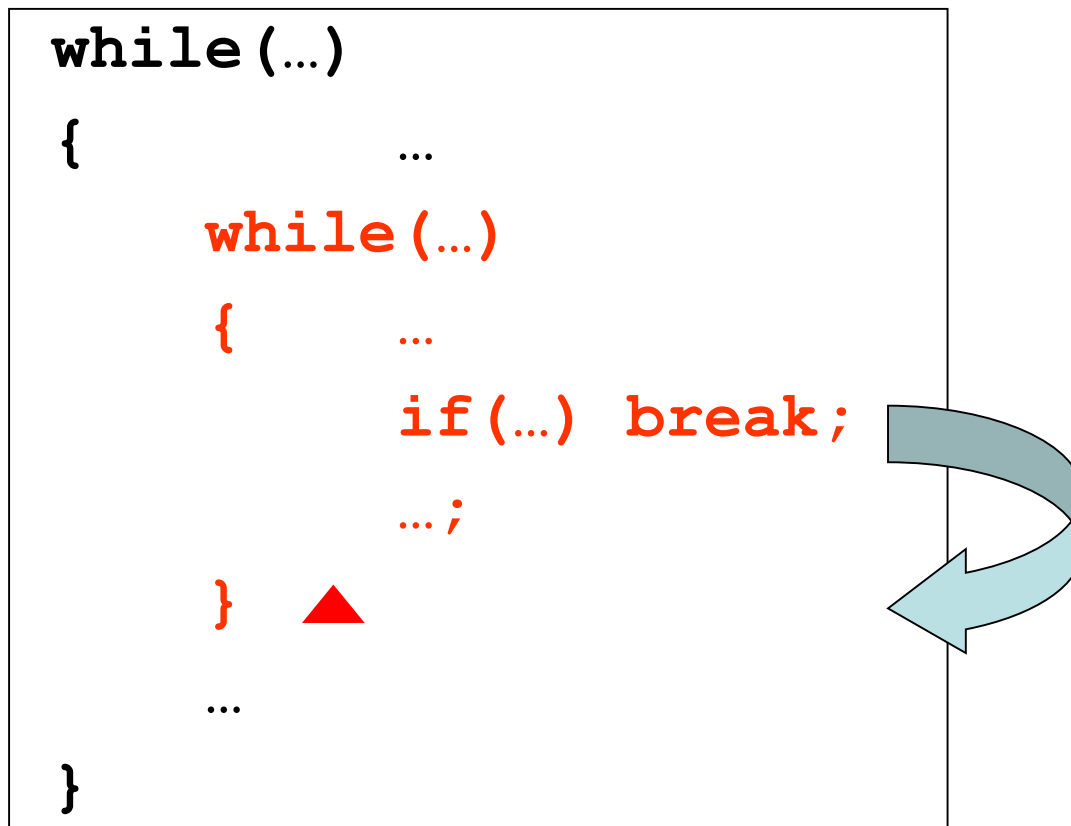
```
int d, sum = 0, i = 1;
while(i <= 10)
{
 scanf("%d", &d);
 if(d <= 0) break;
 sum += d;
 ++i;
} ▲
printf(...
```

```
int d, sum = 0, i = 1;
do
{
 scanf("%d", &d);
 if(d <= 0) break;
 sum += d;
 ++i;
} while(i <= 10); ▲
printf(...
```

```
int d, sum = 0;
for(int i = 1; i <= 10; ++i)
{
 scanf("%d", &d);
 if(d <= 0) break;
 sum += d;
} ▲
printf(...
```

## ❁ 折断的等价形式

- 在嵌套循环中，内层循环体里的break折断内层循环流程。



- 外层循环仍然执行9次，只不过部分内层循环没有执行9次而已。

```
for(int i = 1; i <= 9; ++i)
{
 for(int j = 1; j <= 9; ++j)
 {
 if(i * j > 10) break;
 printf("%d \t", i * j);
 }
 printf("\n");
}
```

//并非一旦乘积超过10就结束程序

|   |    |   |   |    |   |   |   |   |
|---|----|---|---|----|---|---|---|---|
| 1 | 2  | 3 | 4 | 5  | 6 | 7 | 8 | 9 |
| 2 | 4  | 6 | 8 | 10 |   |   |   |   |
| 3 | 6  | 9 |   |    |   |   |   |   |
| 4 | 8  |   |   |    |   |   |   |   |
| 5 | 10 |   |   |    |   |   |   |   |
| 6 |    |   |   |    |   |   |   |   |
| 7 |    |   |   |    |   |   |   |   |
| 8 |    |   |   |    |   |   |   |   |
| 9 |    |   |   |    |   |   |   |   |

---

|   |   |   |   |    |   |   |   |   |
|---|---|---|---|----|---|---|---|---|
| 1 | 2 | 3 | 4 | 5  | 6 | 7 | 8 | 9 |
| 2 | 4 | 6 | 8 | 10 |   |   |   |   |

```
for(int i = 1; i <= 9; ++i)
{
 int j;
 for(j = 1; j <= 9; ++j)
 {
 if(i * j > 10) break;
 printf("%d \t", i * j);
 } ▲
 if(i * j > 10)
 break;
 printf("\n");
} ▲
```

# 可以用goto控制嵌套循环的折断

---

```
for(int i = 1; i <= 9; ++i)
{
 for(int j = 1; j <= 9; ++j)
 {
 if(i * j > 10) goto END;
 printf("%d \t", i * j);
 }
 printf("\n");
}
```

END: ;

- 使用goto语句不能跳过变量的初始化。比如，

```
...
while (...)
{
 while (...)
 if (...) goto LOOP2; // ✗
 ...
}
int y = 10;
LOOP2: ...
```

```
...
if (...) goto LOOP3; // ✗
...
for (...)
{
 int y = 10;
LOOP3: ...
}
...
```

- 尽量不要使用goto语句

- 上述约束
- 破坏程序的结构，隐患
- 可读性差，找标签困难



# 用标志变量辅助实现嵌套循环的折断

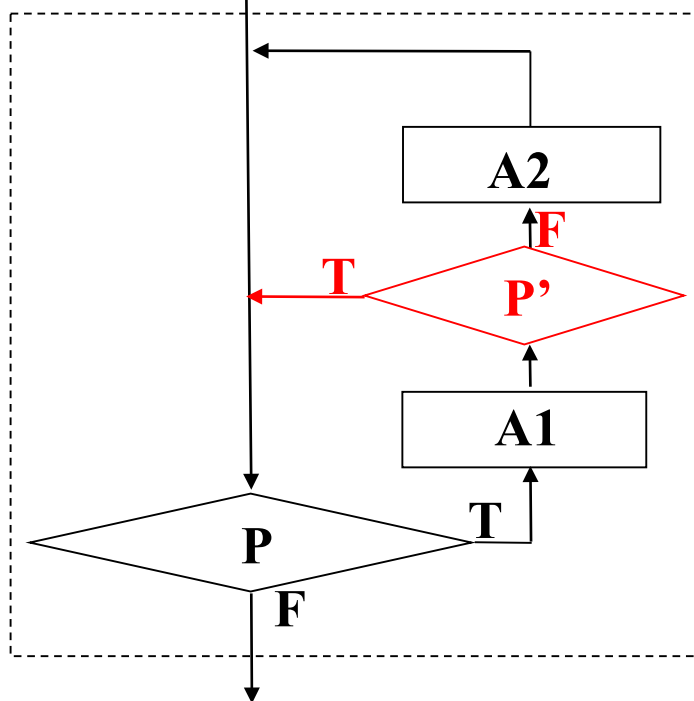
---

```
int flag = 1;
for(int i = 1; i <= 9 && flag; ++i)
{
 for(int j = 1; j <= 9 && flag; ++j)
 {
 if(i * j > 10) flag = 0;
 else printf("%d \t", i * j);
 }
 printf("\n");
}
```

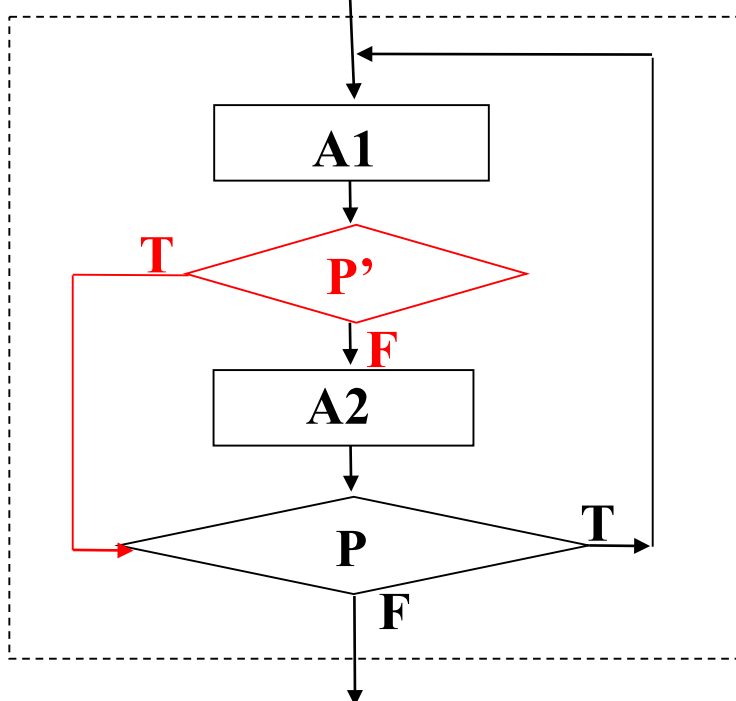
# 循环流程的接续(continue)

- 即循环操作被分成两部分，然后根据一定情况在相应的语句控制下，在执行其中一部分操作后，结束本次循环，提高循环流程的灵活性。

C语言中continue辅助while控制的循环流程 (1)

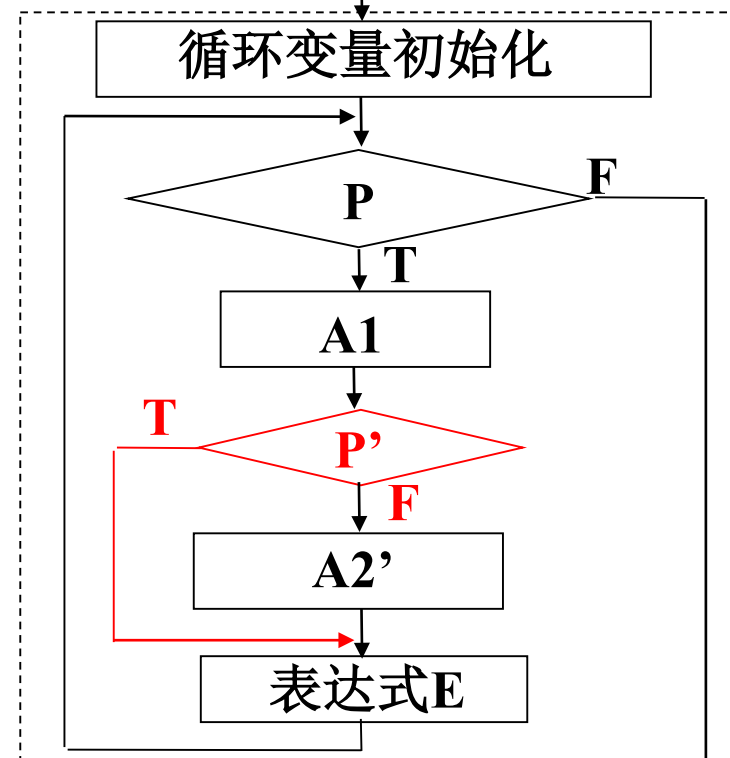


C语言中continue辅助do-while控制的循环流程 (2)



(1)和(2)效果等价

C语言中continue辅助for控制的循环流程 (3)



效果与(1)/(2)不等价

# continue

- 例1.7 设计C程序，求输入的10个正整数的和，遇到负数或0就忽略不计。

```
int d, sum = 0, i = 1;
while(i <= 10)
{
 scanf("%d", &d);
 if(d <= 0) continue; //转下一次循环
 sum += d;
 ++i;
}
printf(...
```

```
1
2
-3
4
5
6
7
8
9
10
11
sum=63
```

循环至少要  
完整地执行  
10次。

```
int d, sum = 0, i = 1;
while(i <= 10)
{
 scanf("%d", &d);
 if(d <= 0) continue;
 sum += d;
 ++i;
}
printf(...
```

等价于:

```
1
2
3
4
5
6
7
8
9
10
11
sum=63
```

```
int d, sum = 0, i = 1;
do
{
 scanf("%d", &d);
 if(d <= 0) continue;
 sum += d;
 ++i;
}
while(i <= 10);
printf(...
```

## continue 辅助 for 控制的流程

```
int d, sum = 0;
for(int i = 1; i <= 10; ++i)
{
 scanf("%d", &d);
 if(d <= 0) continue;
 sum += d;
}
printf(...;
```

```
1
2
-3
4
5
6
7
8
9
10
sum=52
```

# 除非:

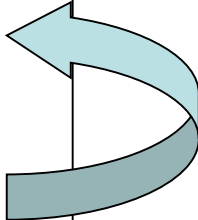
## continue辅助for控制的流程

```
int d, sum = 0;
for(int i = 1; i <= 10;)
{
 scanf("%d", &d);
 if(d <= 0) continue;
 sum += d;
 ++i;
}
printf(...;
```

```
1
2
-3
4
5
6
7
8
9
10
11
sum=63
```

- 在嵌套循环中，内层循环体里的continue接续内层循环流程。

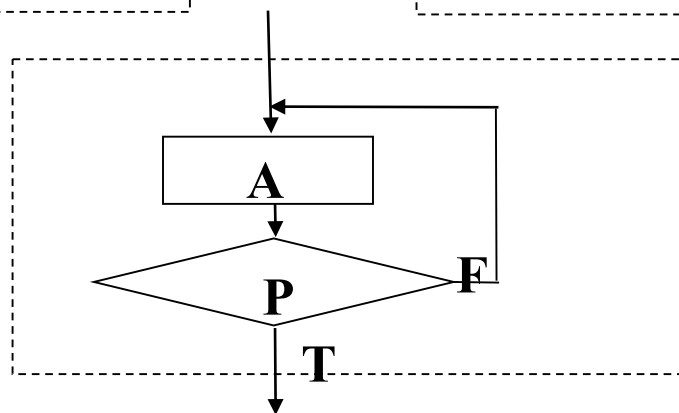
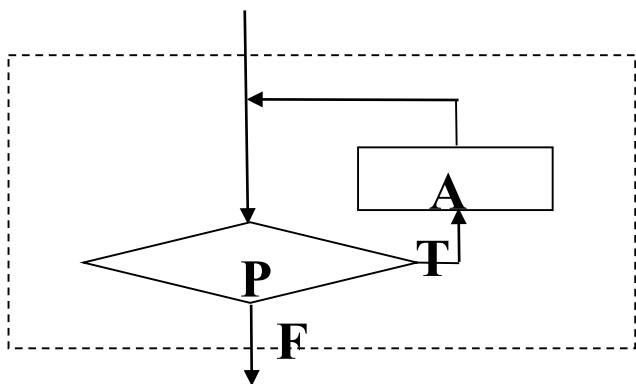
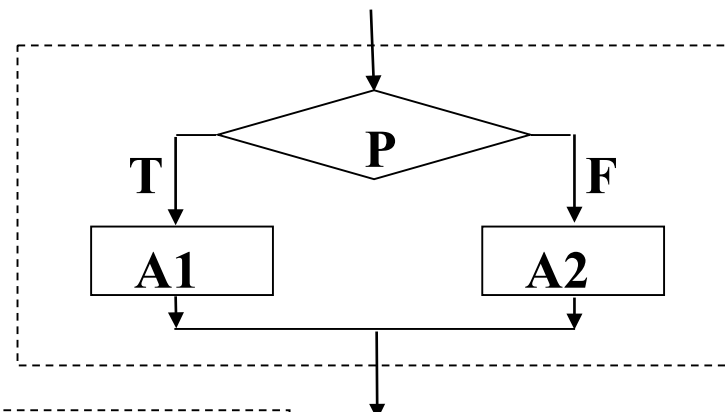
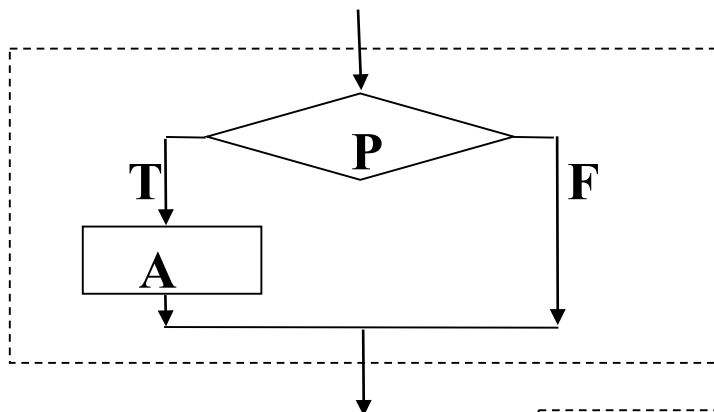
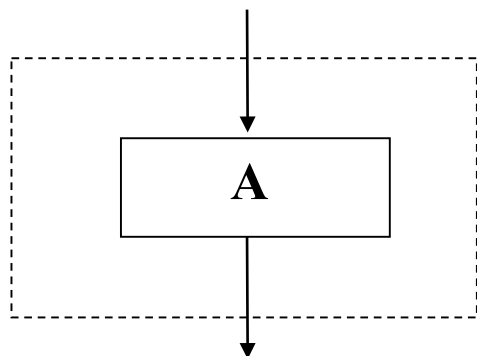
```
while (...)
{
 ...
 while (...)
 {
 ...
 if (...) continue;
 ...;
 }
 ...
}
```



# 流程控制方法小结

## 三种基本流程的共同特点：

- 只有一个入口；
- 只有一个出口；
- 流程内的每一部分都有机会被执行到。





# C语言的流程控制语句 (statement)

## 对应基本流程，C语言提供了控制语句

- if、if-else、switch (有条件的选择语句)

- while、do-while、for (循环语句)

结构化

## 此外，C语言还提供了流程辅助控制语句 (无条件转移语句)

- break

- continue

- goto

半结构化

非结构化

## 函数调用与return语句

## C语言里的其他语句可作为基本流程的子语句

- 复合语句

- 表达式语句

- 空语句

# 小结

---

## 分支流程及其控制方法

➤ if...

conditional clauses

➤ if...else...

alternative clauses

➤ switch... (break)

choice clauses

➤ 嵌套

## 循环流程及其控制方法

repetition clauses

➤ while...

➤ do...while...

➤ for...

➤ 嵌套

➤ break / continue

## 运用

➤ 自顶向下，逐步求精

➤ 分类、穷举、迭代



## 要求：

- ◆ 会运用分支/循环流程控制语句实现简单的计算任务
  - 一个程序代码量 $\approx 20$ 行，  
在main函数中完成数据定义、输入、分支/循环处理、输出
- ◆ 能够定位出错行，修改程序中的语法错误
- ◆ 继续保持良好的编程习惯
  - 不用goto，子语句缩进并写在花括号中，多分支语句代替多个并列的if，循环语句使用原则...

# Thanks!

---

