# 作业

- Ex1. 编程实现：输入两个整数，输出它们的平方和。

```
#include <stdio.h>  // #include <cstdio>
int main()
{     int m, n;
      scanf("%d%d", &m, &n);
      printf("%d \n", m*m + n*n);
      return 0;
}
```

```
#include <iostream>
using namespace std ;
int main()
{     int m, n;
      cin >> m >> n;
      cout << m*m + n*n << endl;
      return 0;
}
```

- Ex2. 编程实现：输入方程的系数a、b、c，计算一元二次方程ax2+bx+c=0的根的判别式，并输出，结果保留2位小数。

```
#include <stdio.h>                                  // #include <iostream>
                                                    // using namespace std;
                                                    // #include <iomanip>
int main()
{
    double a, b, c;
    scanf("%lf%lf%lf", &a, &b, &c);      // cin >> a >> b >> c;
    …
    printf("%.2f \n", …);                           // cout << fixed <<…;
    return 0;
}
```

# 课后任务

- 下列C程序的功能是：每当用户输入一个整数，就输出已输入的所有整数的乘积，共输入三次。修改程序，以便执行程序后可以获得正确的结果。

```
#include <stdio.h>
int main()
{    int p, n;          // int p = 1, n;
     printf("Please input an integer: ");
     scanf("%d", &n);
     p = p * n;         //该程序易于用循环语句改写，故不要修改此行代码
     printf("The product is %d. ", p);
     printf("Please input an integer: ");
     scanf("%d", &n);
     p = p * n;
     printf("The product is %d. ", p);
     ……
```

- 用\n与续行符修改下面的程序，实现用*号输出一个钻石形图案。

```
#include <stdio.h>
int main()
{
    printf("\n\
              ***\n\
            *******\n\
             *****\n\
              ***\n\
               *\n");
    return 0;
}//验证程序中看起来的换行并不对应显示器上输出内容的换行
```

南京大学 计算机科学与技术系
Department of Computer Science & Technology, NJU

# step by step

## 进阶

刘奇志

- 程序的流程控制方法
- 程序的模块设计方法
- 程序中操作的描述
- 程序中数据的描述
- 专题
  - 数组
  - 指针
  - 字符串
  - 结构体
  - 文件

南京大学 计算机科学与技术系
Department of Computer Science & Technology, NJU

# Chapter 1

## 程序的流程控制方法

刘奇志

- **顺序流程**
- **分支流程控制方法**
  - 分支流程的基本形式
  - 分支流程的嵌套
  - C语言其他分支流程控制语句
- **循环流程控制方法**
  - 循环流程的基本形式
  - C语言其他循环流程控制语句
  - 循环流程的嵌套及其优化
  - 循环流程的折断和接续
- **综合运用**

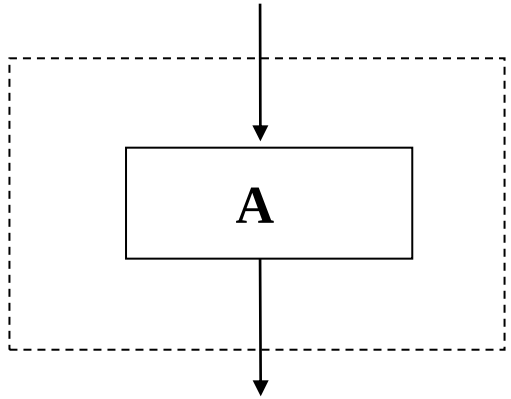
# 流程

- **语句的执行次序**
  - 一般情况下，程序中的语句按照排列次序顺序执行；
  - 如果遇到控制语句，比如C语言中的if语句和while语句等，则会改变执行的顺序；

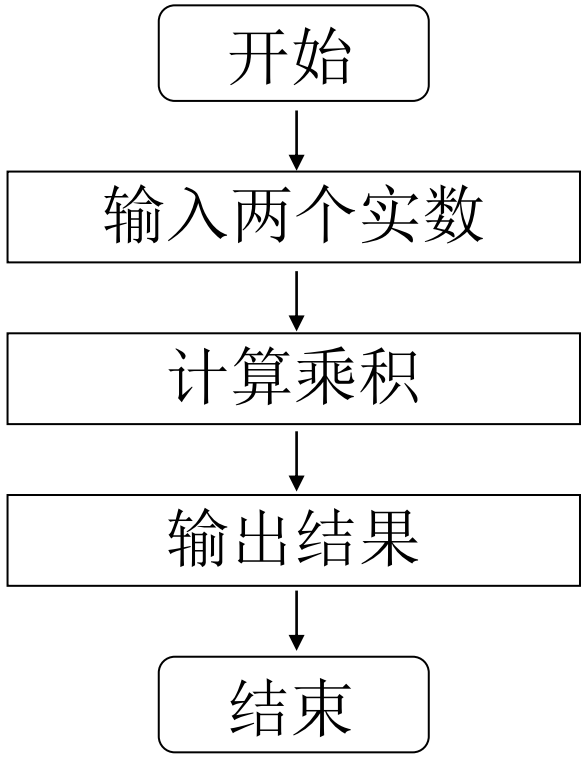
  - 控制语句改变执行次序有分支和循环两种基本方式。
  - 顺序、分支和循环是程序中的三种基本流程。
  - 一个程序的总流程由基本流程衔接而成。

# 顺序流程(sequential flow)

- 顺序流程是没有控制语句控制的流程，即按源程序的语句从上到下逐句执行，每条语句执行一次。



- **编程序，输入两个实数，输出它们的乘积。**



- 顺序流程中，语句的书写次序就是程序的执行次序。
- **语句相同，只是次序稍有不同，结果会输出不同的值：**

  - 片段1：

```
int i = 1, sum = 0;
i = i + 1;
sum = sum + i;
printf("sum = %d \n", sum);          //输出sum = 2
```

  - 片段2：

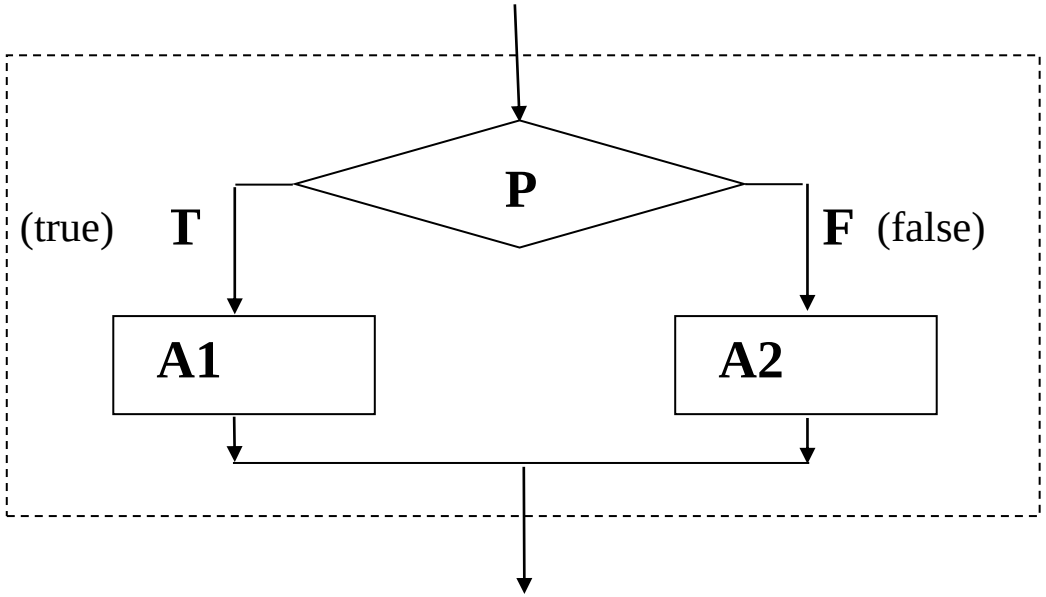
```
int i = 1, sum = 0;
sum = sum + i;
i = i+1;
printf("sum = %d \n", sum);          //输出sum = 1
```

# 分支流程(selection flow) 的基本形式

- 分支流程用于选择性计算场合。

- 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。



- **C语言中的if…else…语句实现这种分支流程的控制**

**if(<条件P>)**

**<if子句>**

**else**

**<else子句>**

- 条件P一般是带有关系或逻辑操作的表达式，表达式的值为true，则认为条件成立，执行if子句，否则执行else子句。

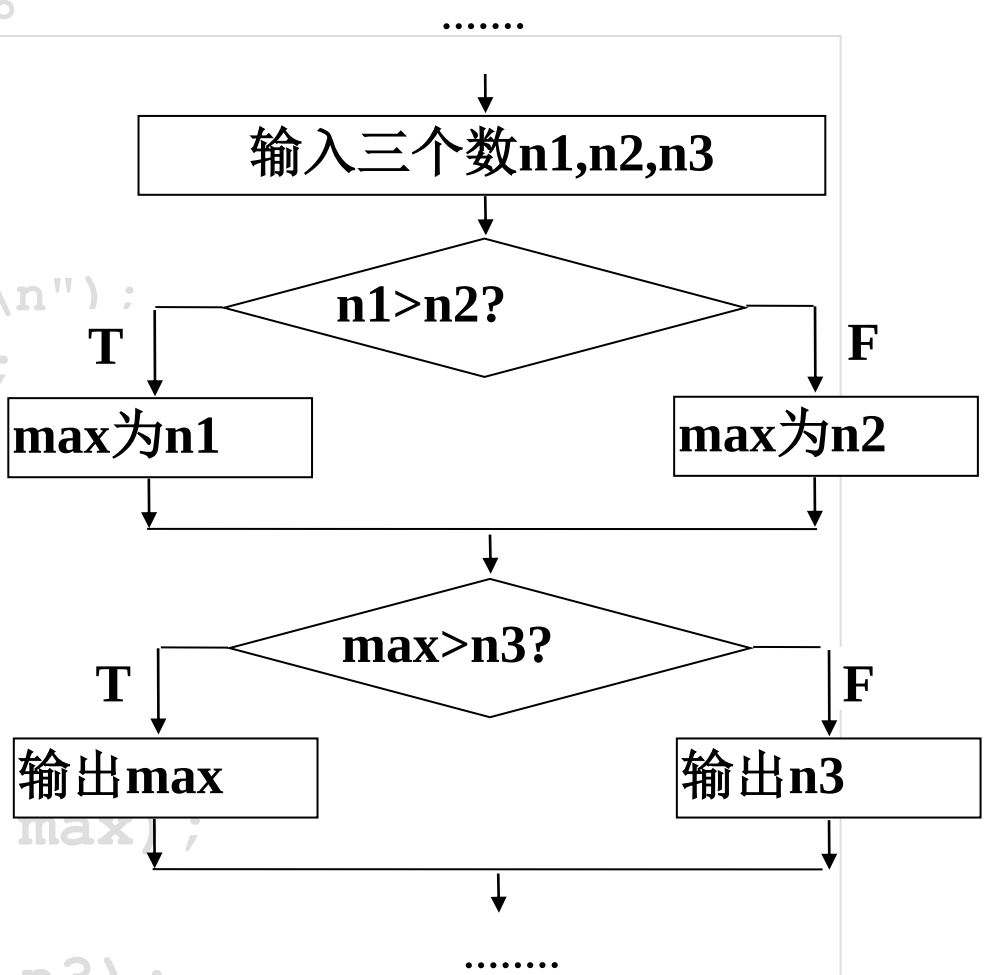
- 关系操作：比如 n > 10
- 逻辑操作：比如 n > 10 && n < 100

## 例1.1 求输入三个整数中的最大值并输出。

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

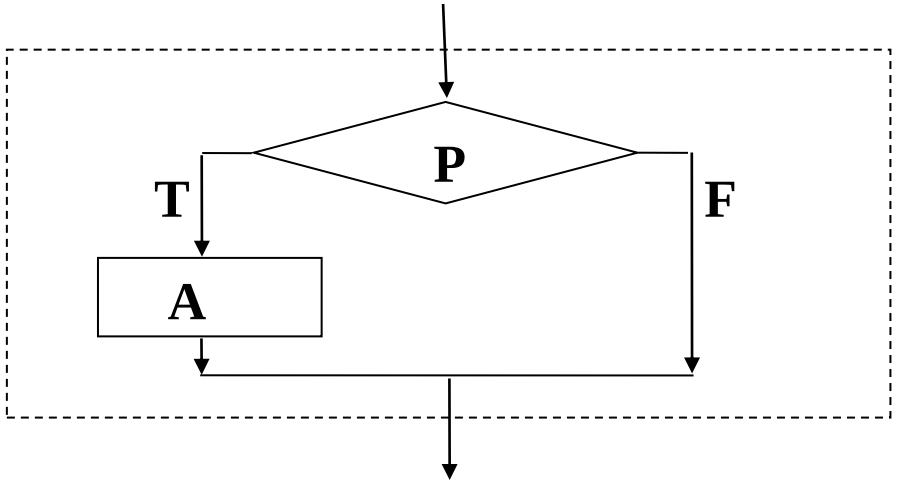
## 例1.1 求输入三个整数中的最大值并输出。

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



# 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。

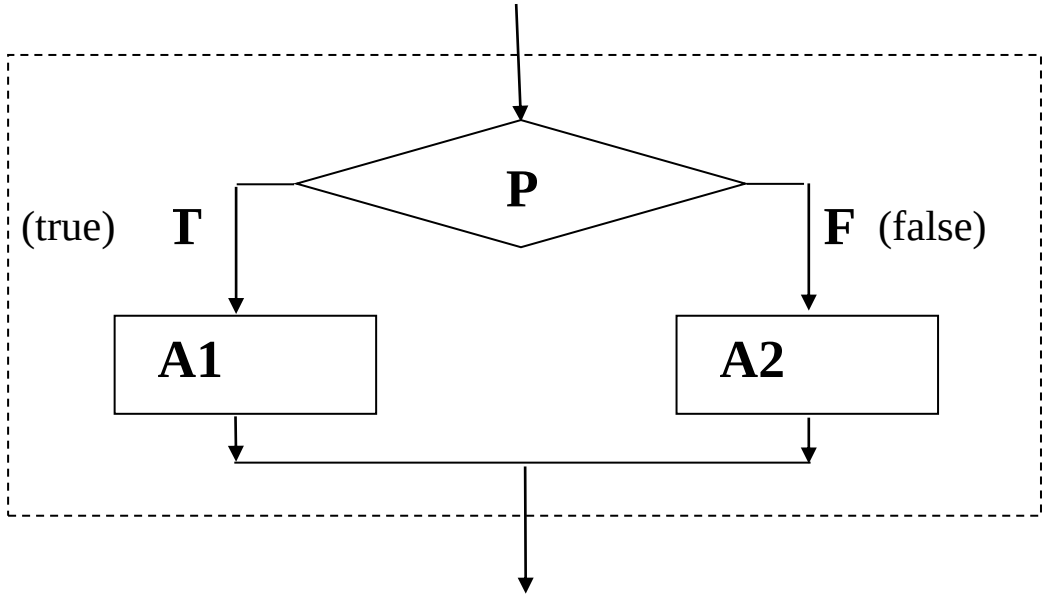


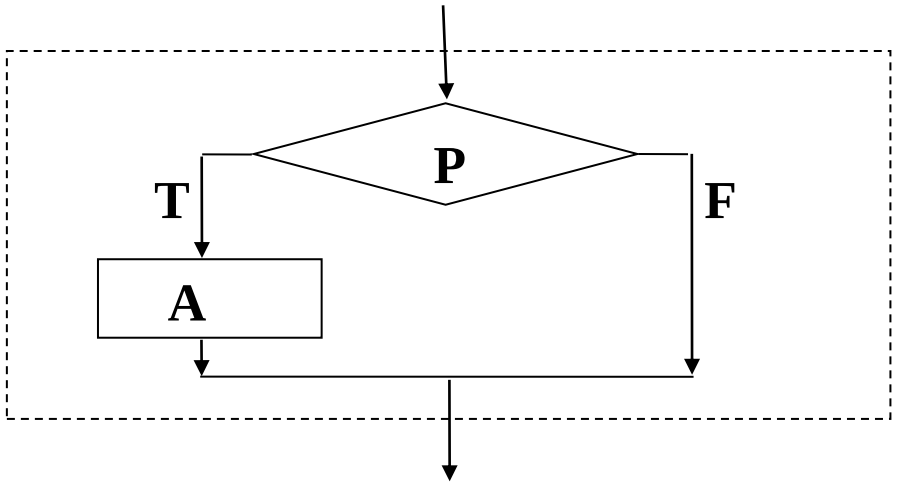
- **C语言中的if…语句实现这种分支流程的控制**

**if(<条件P>)**

**<if子句>**

- 条件P一般是带有关系或逻辑操作的表达式，表达式的值为true，则认为条件成立，执行if子句，否则不执行任何子句。



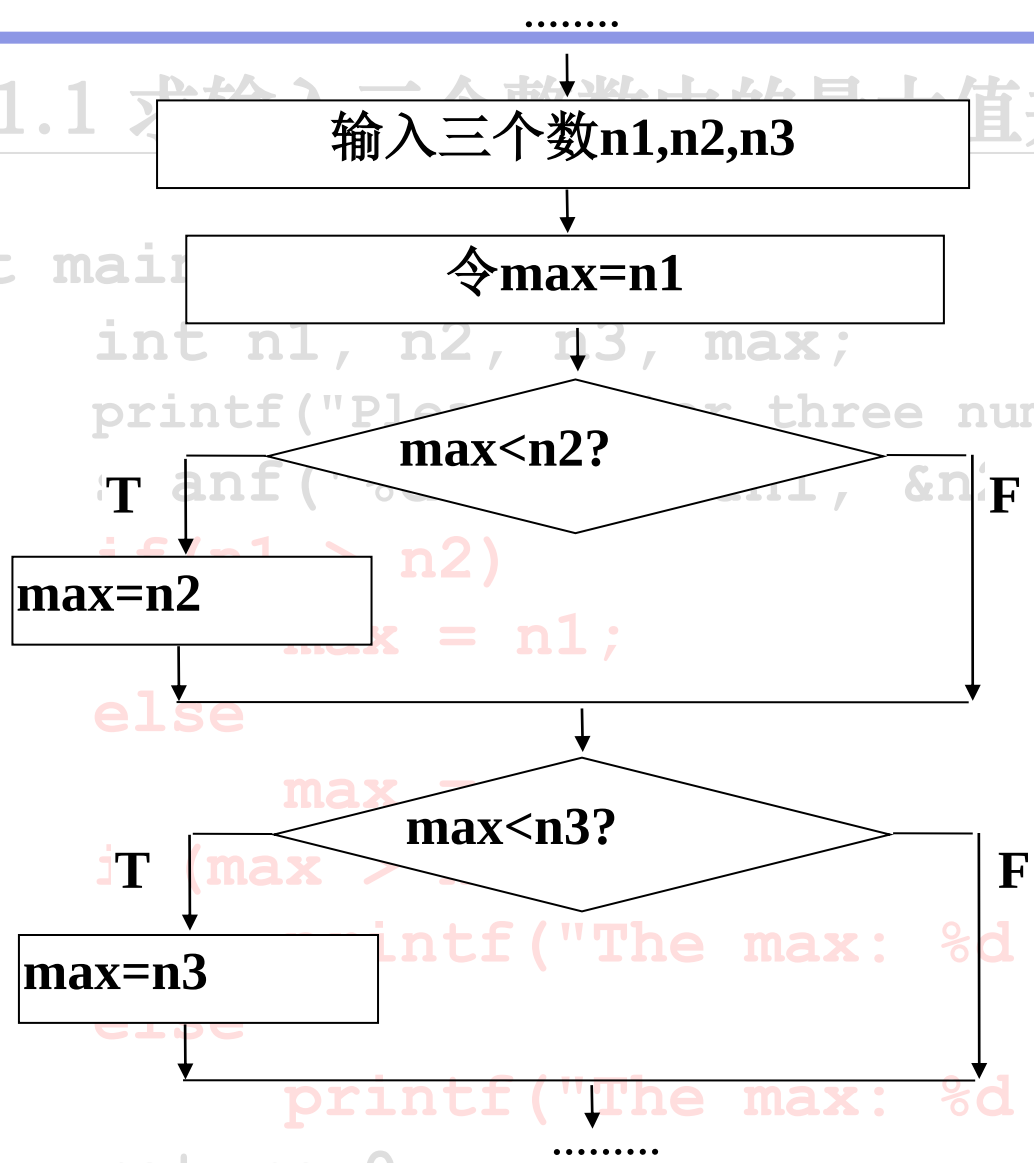

- **分支流程中的条件只判断一次，每个任务最多只执行一次。**

## 例1.1 求输入三个整数中的最大值并输出。

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

```
max = n1;
if(max < n2)
   max = n2;
if(max < n3)
   max = n3;
...//输出max
```

例1.1 求输入三个整数中的最大值并输出。




```
max = n1;
if(max < n2)
    max = n2;
if(max < n3)
    max = n3;
...//输出max
```

# 分支流程的书写

- **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");
```

# 分支流程的嵌套

- 多个分支流程可以嵌套，成为多分支形式。

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

```
if(n1 > n2)
      if(n1 > n3)
            max = n1;
      else
            max = n3;
else
      if(n2 > n3)
            max = n2;
      else
            max = n3;
...//输出max
```

- 在编辑嵌套的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，缩进会使程序正文过分偏右，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");
```

- 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
……
```

- 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。

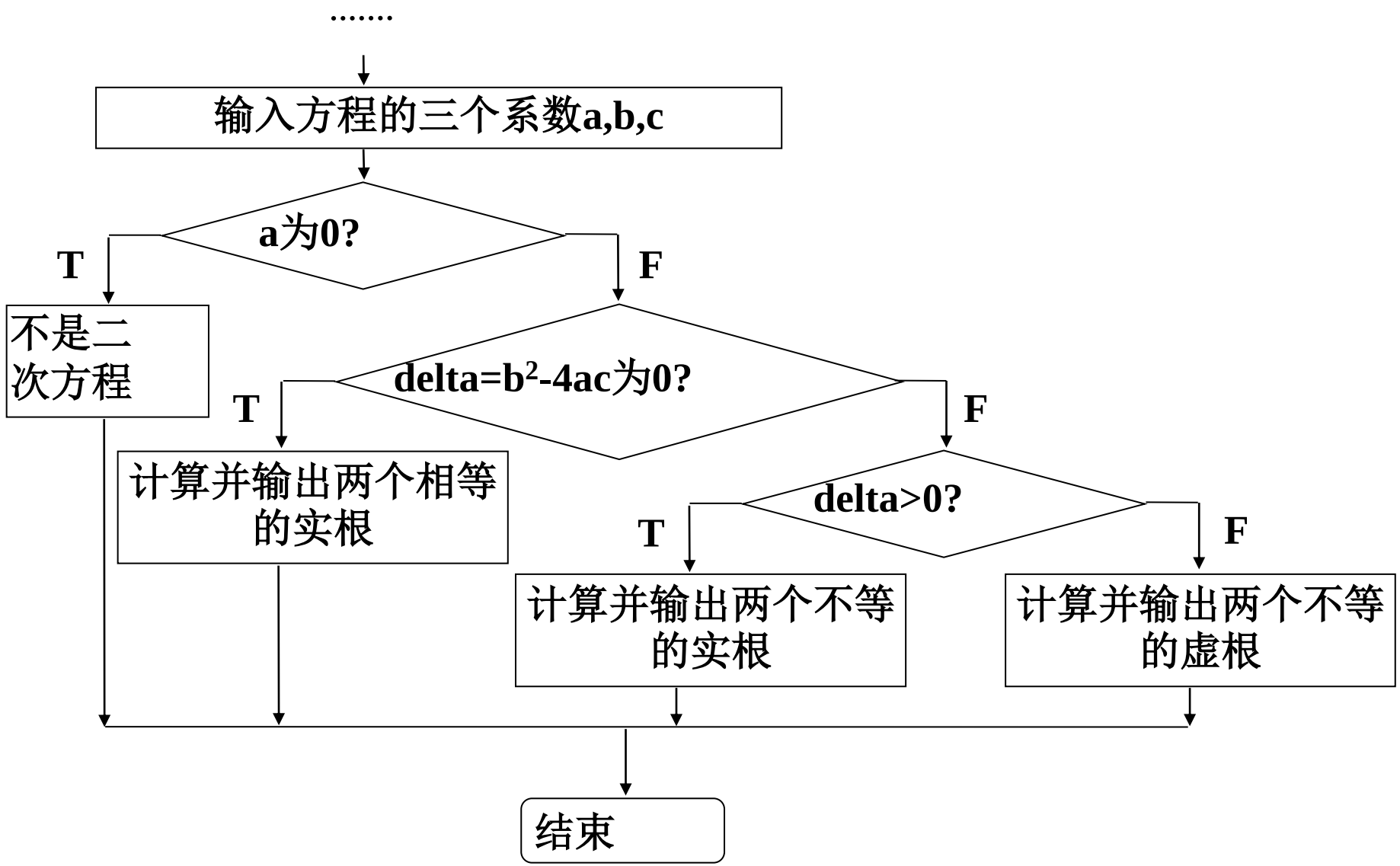
- **如果在逻辑上需要将else子句与较远的if子句配对**
  - 可以用一个花括号把较近的if子句写成复合语句

```
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子句后面构造一个用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*/
```

- 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 <cmath>
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，本行之前没有空格
       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
```

- **嵌套的分支流程往往能够避免不必要的条件判断，比如：**

```
if(score >= 90)
 printf("优");
if(score >= 80 && score < 90)
 printf("良");
……
if(score < 60)
 printf("不及格");
```

```
if(score >= 90)
  printf("优");
else if(score >= 80)
  printf("良");
……
else if(score >= 60)
  printf("及格");
else
  printf("不及格");
```

- **改写成嵌套形式，在 score >= 90 时可以避免后面多个条件判断**

# C语言其他分支流程控制语句- switch语句

- C语言还提供了一种叫做开关语句的switch语句，能实现部分多分支流程的控制。

```
switch(grade)       //该行没有分号
{
      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");
}                   //该行没有分号
```

```
char grade;
grade = getchar();
```

```
switch(week)        //该行没有分号
{
      case 0: printf("Sunday \n"); break;
      case 1: printf("Monday \n"); break;
      ……
      default: printf("error \n");
}                   //该行没有分号
```

```
int week;
scanf("%d", &week);
```

- **switch后面圆括号中的操作结果，与case后面的整数或字符常量进行匹配**
  - 如果匹配成功，则从冒号后的语句开始执行，执行到右花括号结束该流程； 如果没有匹配的值，则执行default后面的语句或不执行任何语句（default分支可省略），然后结束该流程。
- **每个case后面的语句最多执行一次。**
- **必须保证case后面的值各不相同，否则无法进行匹配。**

```
 if(grade == 'A')
     printf("90-100 \n ");
 else if(grade == 'B')
     printf("80-89 \n ");
 else if(grade == 'C')
     printf("70-79 \n ");
…
 else
     printf("error \n ");
```

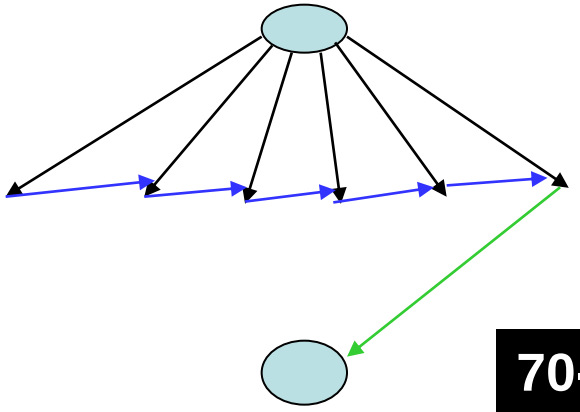
对于能够使用switch语句控制的多分支流程，使用switch语句往往便于编译器进行优化， 以便获得比if语句更高的效率。

# 运用switch语句的注意事项：

- switch语句中的case仅仅与紧随其后的整数或字符常量一起作为语句标号，没有流程控制作用
- “break;”语句具有流程控制作用，它将流程转向switch 语句结束处
- 没有“break;”则执行完本分支任务后，会继续执行其他分支的任务，不管其他分支case后的整数或字符常量是否匹配

‘C’

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



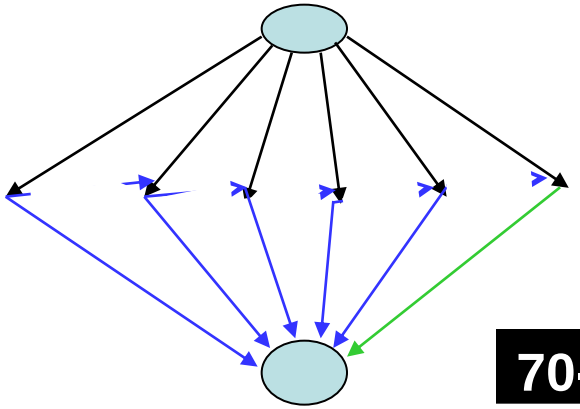
70-79
60-69
0-59
error

# 运用switch语句的注意事项：

- switch语句中的case仅仅与紧随其后的整数或字符常量一起作为语句标号，没有流程控制作用
- “break;”语句具有流程控制作用，它将流程转向switch 语句结束处
- 没有“break;”则执行完本分支任务后，会继续执行其他分支的任务，不管其他分支case后的整数或字符常量是否匹配

‘C’

```
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;
}
```



70-79

- 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语句可以嵌套，这时，内层switch语句里的“break;”语句只能将程序的流程转向内层switch语句的结束处，不能控制外层switch语句的流程。

```
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
}
```

# C语言其他分支流程控制语句- goto语句

- C语言还保留了goto语句。goto语句与if语句以及后面某个语句的标号配合使用也能实现分支流程的控制

```
        max = n1;
        if(n2 < max)
                goto T1;
        max = n2;
T1:     if(max < n3)
                max = n3;
```

- 但这类流程完全可以不用goto语句，只要改写goto语句所在的if语句条件即可，比如上面的程序片段可以改写成：

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

- 由于goto语句容易使程序流程混乱，尽量不要用goto语句实现分支流程的控制。

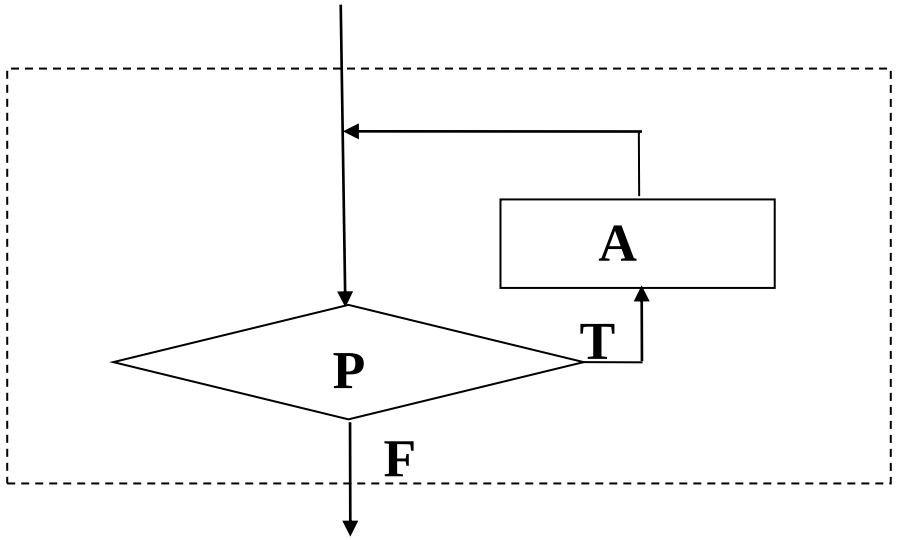
- 顺序流程
- 分支流程控制方法
  - 分支流程的基本形式
  - 分支流程的嵌套
  - C 语言其他分支流程控制语句
- 循环流程控制方法
  - 循环流程的基本形式
  - C 语言其他循环流程控制语句
  - 循环流程的嵌套及其优化
  - 循环流程的折断和接续
- 综合运用

# 循环流程

- 计算机在完成一个任务时，常常需要对相同的操作重复执行多次（每次操作数的值可能有所不同）
- 循环流程（iteration flow，loop flow）用于这种重复性计算场合，它是程序中最重要的一种流程
- 循环流程由循环语句控制

# 循环流程的基本形式

- **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。



- **C 语言中的 while 语句可以实现这种循环流程的控制，其格式为：**

**while(<条件P>)**

**<任务A>**

  - 任务A是while语句的子句，通常是一个复合语句，可以写在右圆括号的后面，最好写在下一行，并缩进。
  - 条件P一般是带有关系或逻辑操作的表达式，表达式的值为true，则认为条件成立，执行任务A，并再次判断条件P，否则不执行任务A，也不再判断条件P 。

## 例1.3 求N（比如100）以内自然数的和，并输出。

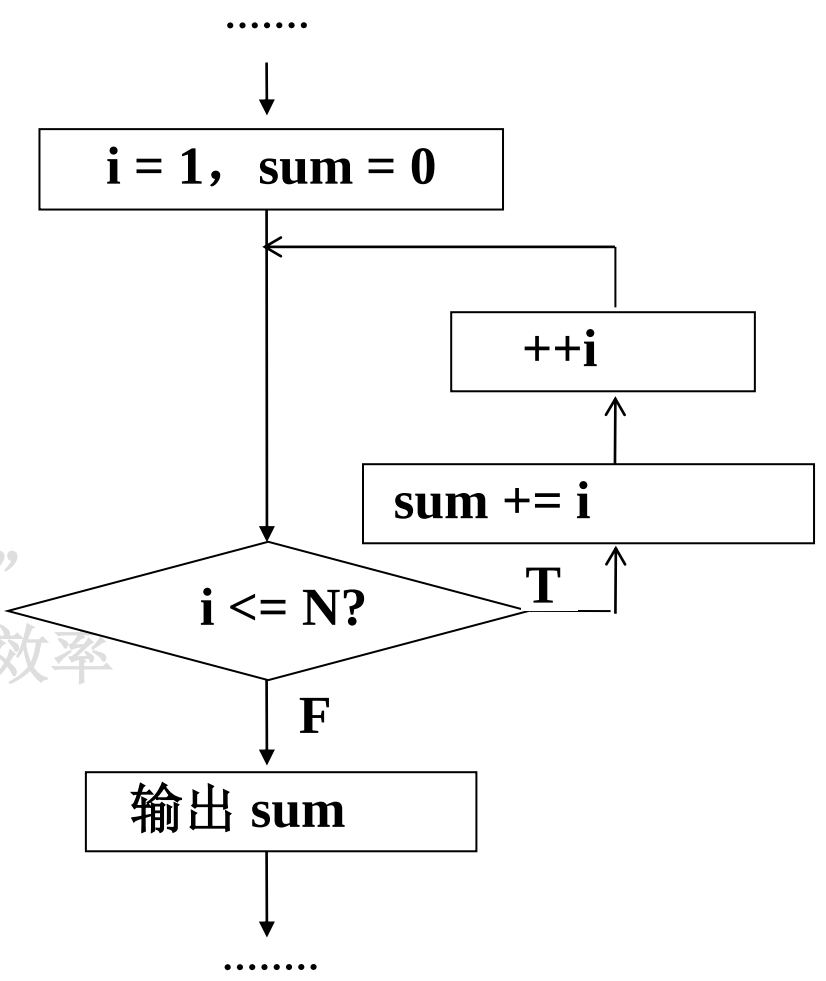
```
…
#define N 100
int main()
{
      int i = 1, sum = 0;
      while(i <= N)          //该行没有分号
      {
            sum = sum + i; //可改为“sum += i;”
            i = i + 1;       //可改为“++i;”提高效率
      }
      printf("Sum. of integers 1-%d: %d\n", N, sum);
      return 0;
}
```

--i 可以代替 i = i - 1

```
Sum. of integers 1-100: 5050
```

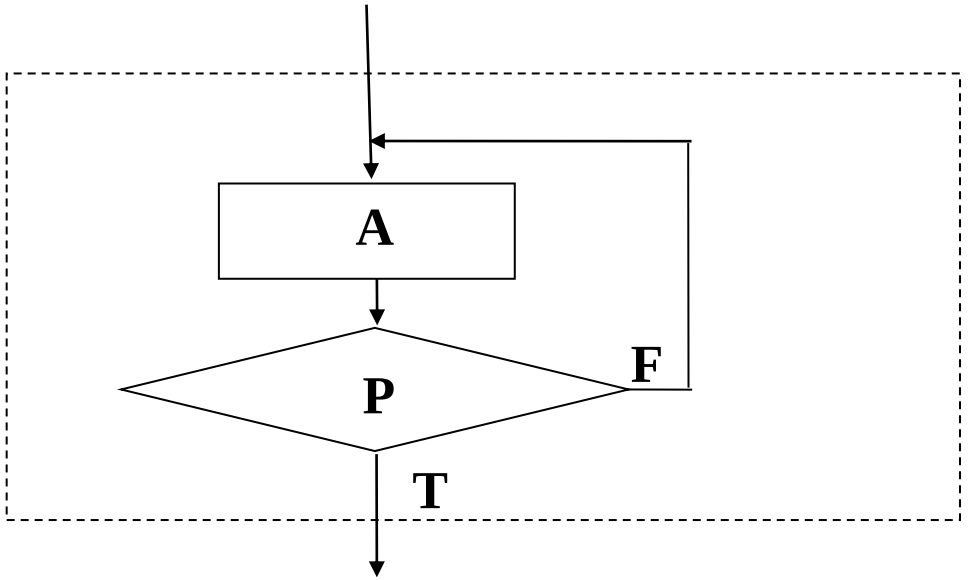
## 例1.3 求N（比如100）以内自然数的和，并输出。

```
…
#define N 100
int main()
{
    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;”提高效率
    }
    printf("Sum. of integers 1-%d: %d\n", N, sum);
    return 0;
}
```



## 循环流程的另外一种形式,包含一个任务和一个条件判断

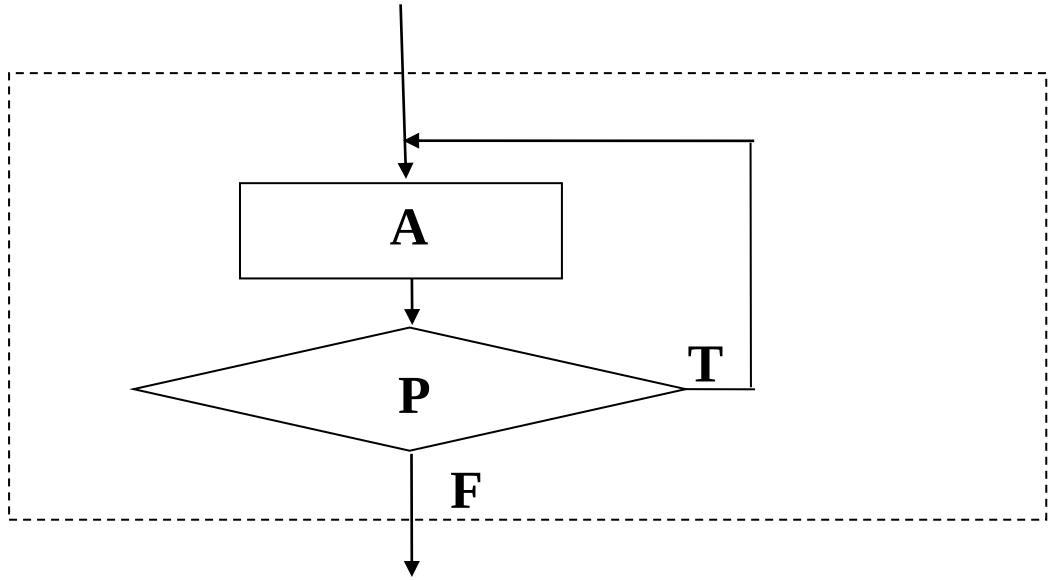
- 先执行一次任务A，再判断条件P
  - 当条件P不成立时继续执行任务A，并再次判断条件P，如此循环往复；
  - 当条件P成立时，该流程结束。



- **C语言中的do-while语句可以实现这种循环流程的控制（类似），其格式为：**

```
do
{
    <任务A>
}while(<条件P>);
```

C语言do-while语句控制的流程




- 任务A是do-while语句的子句，是一个复合语句，写在do的下一行，并缩进。
- 条件P一般是带有关系或逻辑操作的表达式，表达式的值为true，则认为条件成立，执行任务A，并再次判断条件P，否则不执行任务A，也不再判断条件P。

## 例1.3’ 求N（比如100）以内自然数的和，并输出。

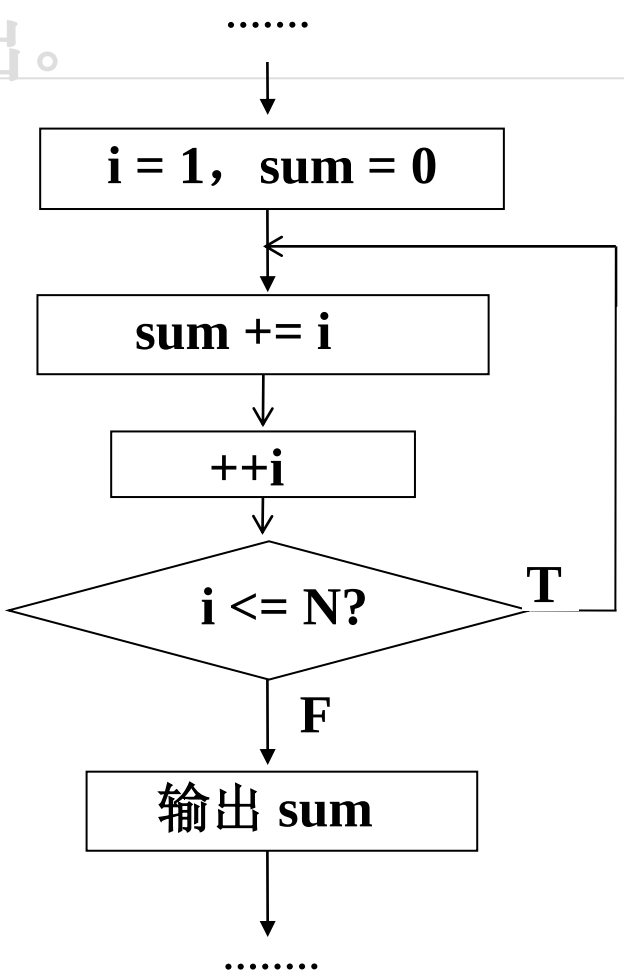
```
…
#define N 100
int main()
{
      int i = 1, sum = 0;
      do
      {
            sum += i;
            i ++;
      }while(i <= N);       //该行有分号

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

## 例1.3’ 求N（比如100）以内自然数的和，并输出。

```
…
#define N 100
int main()
{
    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;
}
```



# 两类循环的区别

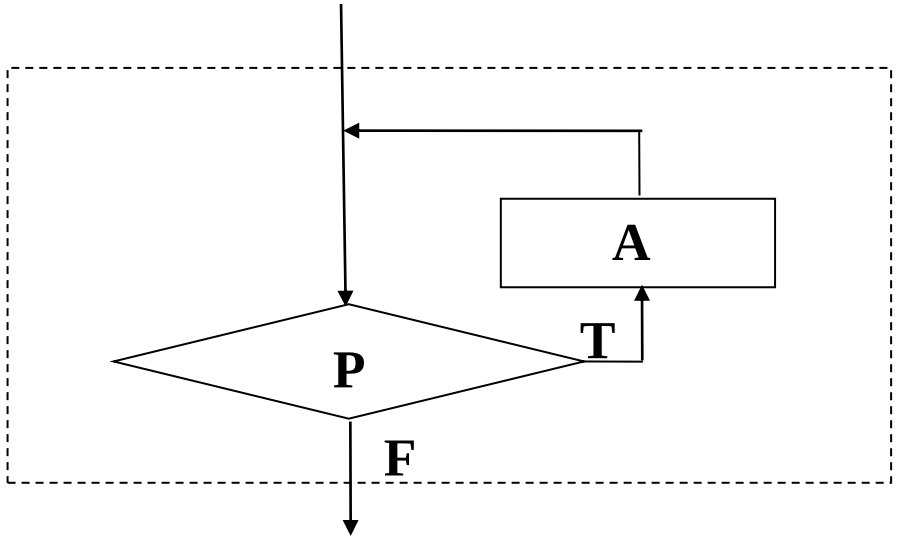
//若 i=101，结果？

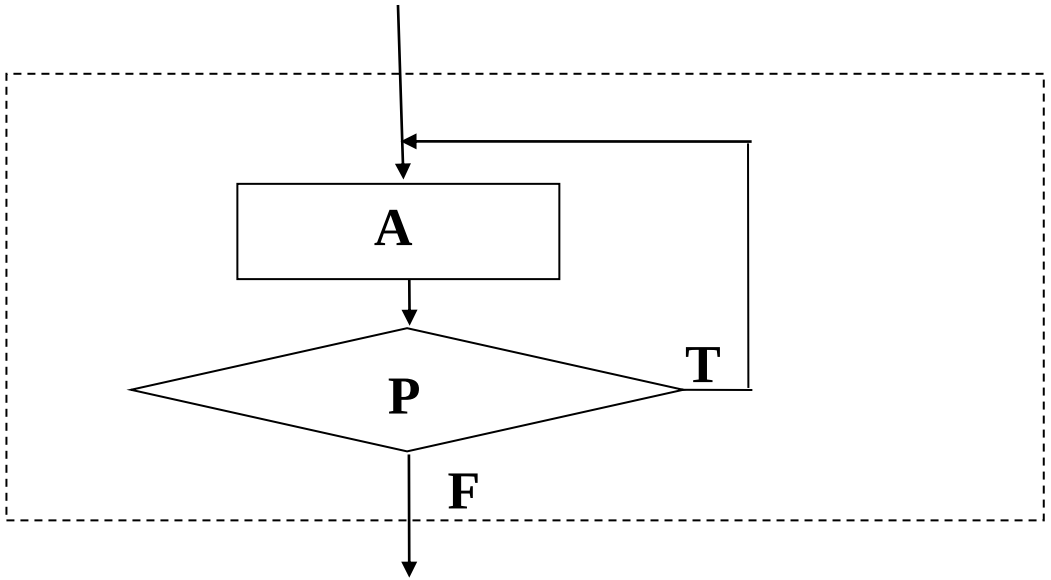
```
int i=1, sum=0;
while(i <= 100)
{
      sum += i;
      ++i;
}
```

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

- 当while语句首次判断表达式的值为真时，与do-while语句效果相同，否则不同。
- i通常称作循环变量，该变量是循环条件判断的依据，改变其值是循环流程控制的关键，进入循环流程前，要先对循环变量进行初始化。

# 两类循环的共同点




- 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，而任务A可能执行有限次（条件P一开始成立，后来不成立），也可能一次不执行/只执行一次（条件P一开始就不成立），甚至可能执行无限次，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。**

# 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。

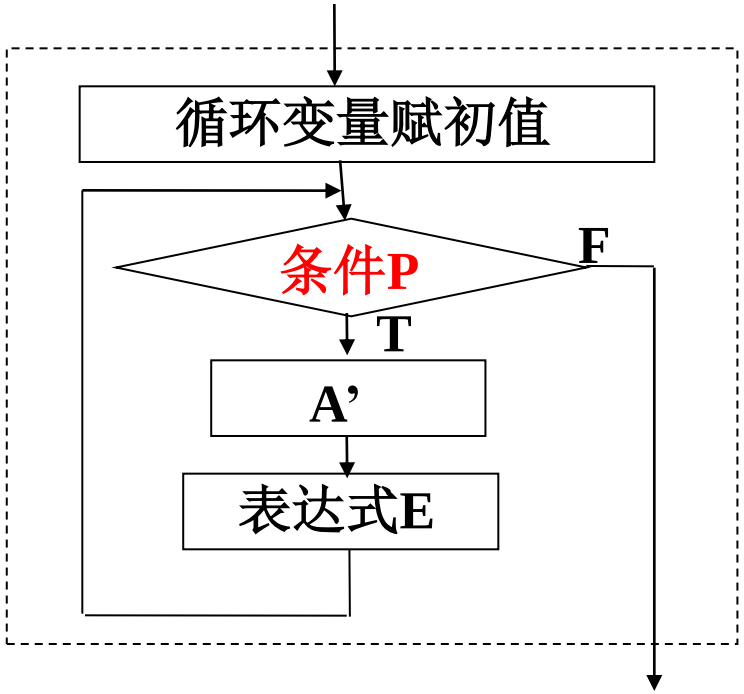
```
while(i <= N)
{
        sum += i;
        ++i;
}
```

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

# C语言其他循环流程控制语句- for语句

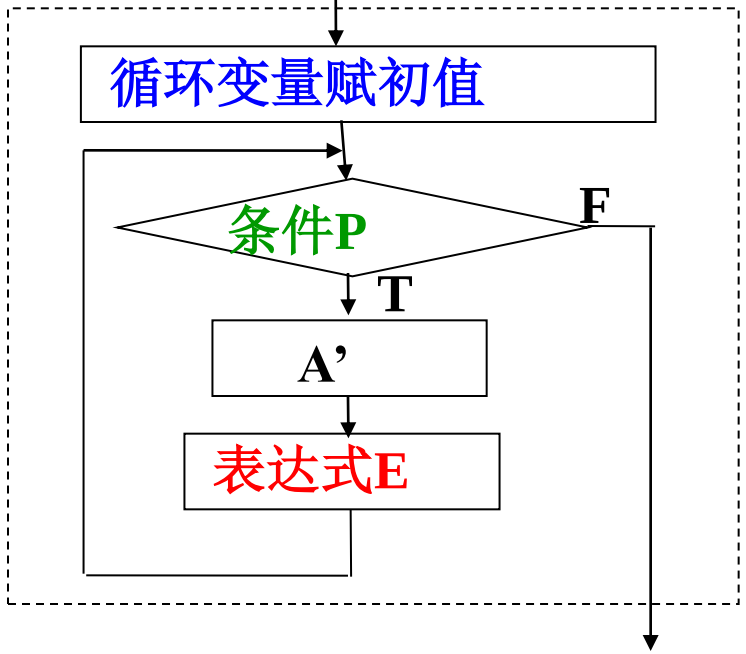
- C语言提供了for语句，也能实现循环流程的控制。

```
for (<循环变量赋初值>;<条件P>;<表达式E>)
{
        <任务A’>
}
```

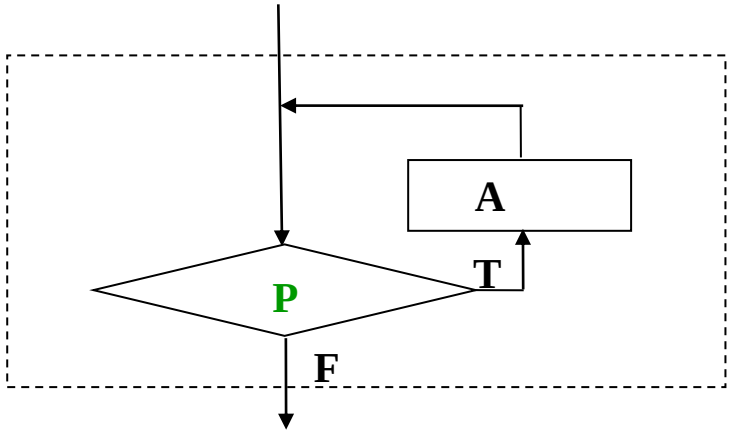



- 先对循环变量赋初值，
  再判断条件P
  - 当条件P成立时，执行任务A’，并计算表达式E，
    然后再判断条件，如此循环往复；
  - 当条件P不成立时，结束该流程。

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




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




- for语句一般将循环变量放入循环语句内赋初值（while或do…while语句的循环变量通常在循环语句前赋初值）；表达式E一般是对循环变量的操作，往往称作步长，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。

- for语句容易误写成

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

空语句

导致sum为101

- 可用来延时

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

调整数值，可调整延时时长

  - 如果要延长比较长的时间，可用嵌套的循环

- 如果任务A’含有多条语句，则一定要用花括号将多条语句组合成复合语句，否则，只有第一条语句能被循环执行，从而造成执行结果错误。

```
int i, sum=0;
for(i=1; i <= 100; ++i )
{
      sum += i;
      printf("%d ", i);
}
```

- 单词for后面的圆括号里两个分号必须有（否则会出现语法错误），圆括号里分号前后的内容可以没有（写在其他地方，不过这类写法使for语句本来的优势丢失了），比如，

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

或:

```
int i = 1, sum = 0;
for( ; i <= N; )
{
      sum += i;
      ++i;
}
```

- 有些编译器允许在for语句圆括号里定义循环变量，并且该循环变量往往只在for语句里有效，比如，

```
…
int sum = 0;
for(int i=1; i <= N; ++i)
      sum += i;
printf("%d \n", i); //编译出错
```

- 少数编译器（如VC6.0）在for语句圆括号里第一个分号前定义的变量在for语句外仍然有效。
  - 上述代码在VC6.0下不出错

# C语言其他循环流程控制语句- goto语句

- C语言里保留的goto语句，与if语句以及前面某个语句的标号配合使用也能实现循环流程的控制。

```
      int i=1, sum=0;
T2:   sum += i;
      ++i;
      if(i <= N)   goto T2;
```

- 但这类流程完全可以不用goto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。

- 下面例1.4程序执行后，只能完成一个数的阶乘计算和输出。
- 如果希望在输出一个数的阶乘后程序不结束执行，而是继续等待输入下一个数，并计算和输出下一个数的阶乘，直到用户输入一个结束标志为止，那么，需要再用一个循环流程，重复执行例1.4中的主要语句。修改的程序如例1.5所示。

# 例1.4 求输入的一个正整数的阶乘并输出。

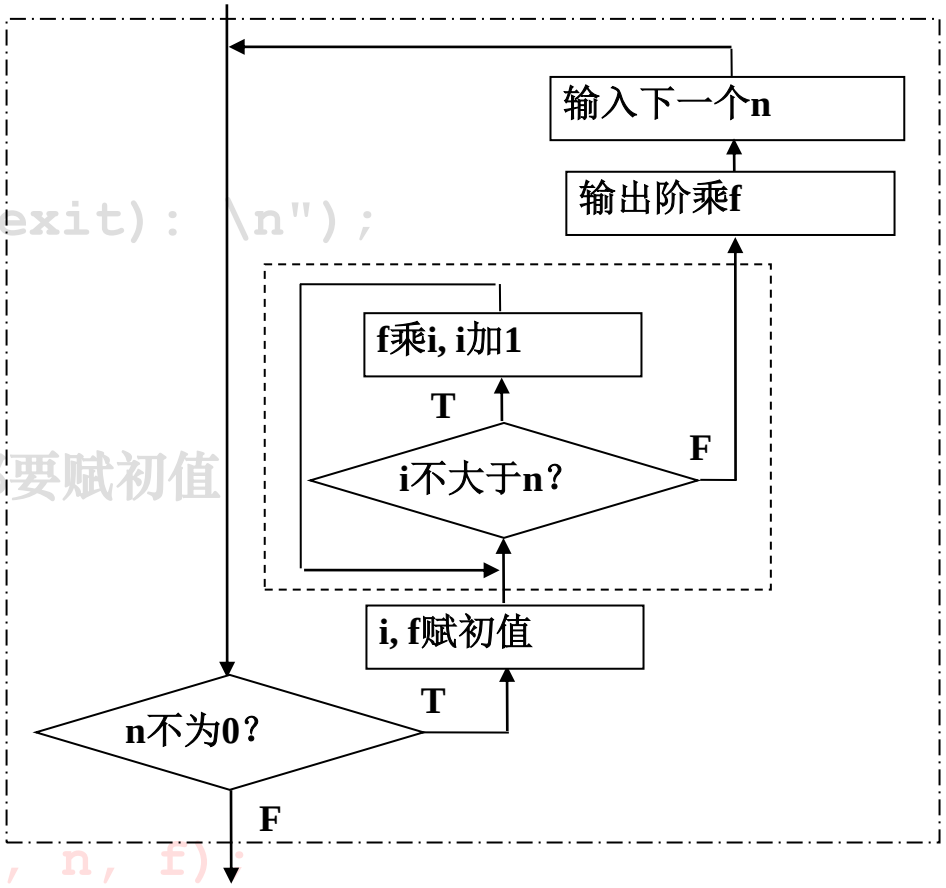
```
…
int main()
{
    int n, i = 2, f = 1; //f 要初始化!
    printf("Please input an integer: \n");
    scanf("%d", &n);
    while(i <= n)
    {
        f *= i;     //相当于f = f * i;
        ++i;
    }
    printf("factorial of %d is: %d \n", n, f);
    return 0;
}
```

# 例1.5 编程实现：每输入一个正整数，输出其阶乘，直到输入0。

```
…
int main()
{       int n, i, f;
        printf("Please input an integer (0 to exit): \n");
        scanf("%d", &n);
        while(n != 0)       // != 表示“不等于”
        {
            i = 2, f = 1; //计算每一个数的阶乘前都要赋初值
            while(i <= n)
            {
              f *= i;
              ++i;
            }
            printf("factorial of %d is: %d \n", n, f);
            printf("Please input another integer (0 to exit): \n");
            scanf("%d", &n);
        }
        return 0;
}
```

# 例1.5 编程实现：每输入一个正整数，输出其阶乘，直到输入0。

```
…
int main()
{       int n, i, f;
        printf("Please input an integer (0 to exit): \n");
        scanf("%d", &n);
        while(n != 0)          // != 表示“不等于”
        {
            i = 2, f = 1; //计算每一个数的阶乘前都要赋初值
            while(i <= n)
            {
              f *= i;
              ++i;
            }
            printf("factorial of %d is: %d \n", n, f);
            printf("Please input another integer (0 to exit): \n");
            scanf("%d", &n);
        }
        return 0;
}
```



- 如果在外层循环外部赋初值，那么在计算和输出完第一次输入的数的阶乘后，i的值不再为2，f的值不再为1，以后的计算结果就不正确了。

```
i = 2, f = 1;       //仅对第一个数的阶乘计算赋了初值
while(n != 0)
{
        while(i <= n)
        {       f *= i;
                ++i;
        }
        ……
```

- 在编辑嵌套的循环语句时，更应采用复合语句和结构清晰的缩进格式，便于发现逻辑错误。

- 例1.5程序中的嵌套循环流程可以用do…while语句或for语句实现。

```
for( ; n != 0; )
{
      for(i = 2, f = 1; i <= n; ++i)
      {
            f *= i;
      }     //循环体只有一条语句时，花括号也可以不加
......
```

- 外循环是事件控制型循环（event-controlled loop），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（counter-controlled loop），其循环条件是计数变量i是否达到边界值n。(for更适合)

# 例1.6 输出一个九九乘法表。

…

```
int main()
{
    printf("          Multiplication Table \n");
    ……
```

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 |

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

制表符

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

- i是外层循环的循环变量，从1递增到9，对应9次循环，每次循环都执行循环体里的两条语句；其中，第一条语句又是一个循环，即内层循环，j是内层循环的循环变量，从1递增到9，对应9次循环，每次循环都执行循环体里的一条语句；整个程序执行完毕时，内层循环的语句执行了81次。

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

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

- 等价的嵌套循环形式

```
……
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;
}
```

```
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");
}
```

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 |

# 并列的循环流程

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

# 嵌套的循环流程

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

# 循环的优化

- 循环操作常常比较耗时，所以编译器往往会做一些优化，以便提高程序的运行效率。例如，

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

上面这种重复计算会被编译器直接优化成：

```
s = a+b;
```

- 循环流程里可以嵌套分支流程。

```
for(i = 0; i < N; ++i)
{       if(…)
                A1;
        else
                A2;
}       //if…else…是一条语句，这对花括号不加也可以，加上更清晰
```

不仅要重复执行分支流程的条件判断，而且由于总是进行条件判断，打断了循环“流水线”作业，使编译器不能对循环进行优化处理，降低了效率。

- 如果循环次数很大，可以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;
```

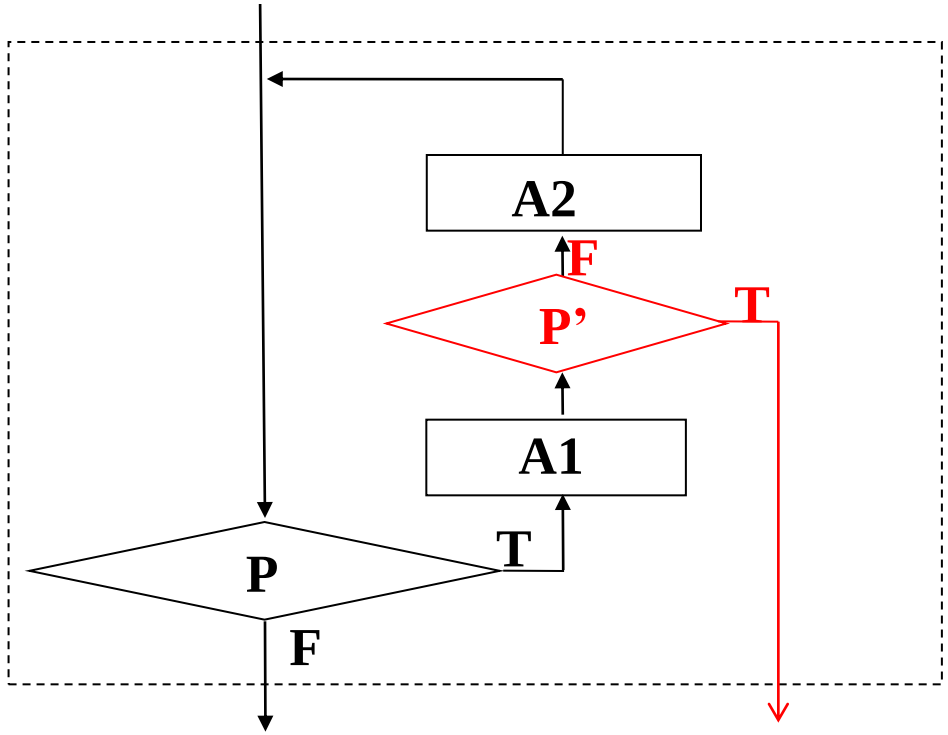
当N很大时可提高效率，但程序不简洁

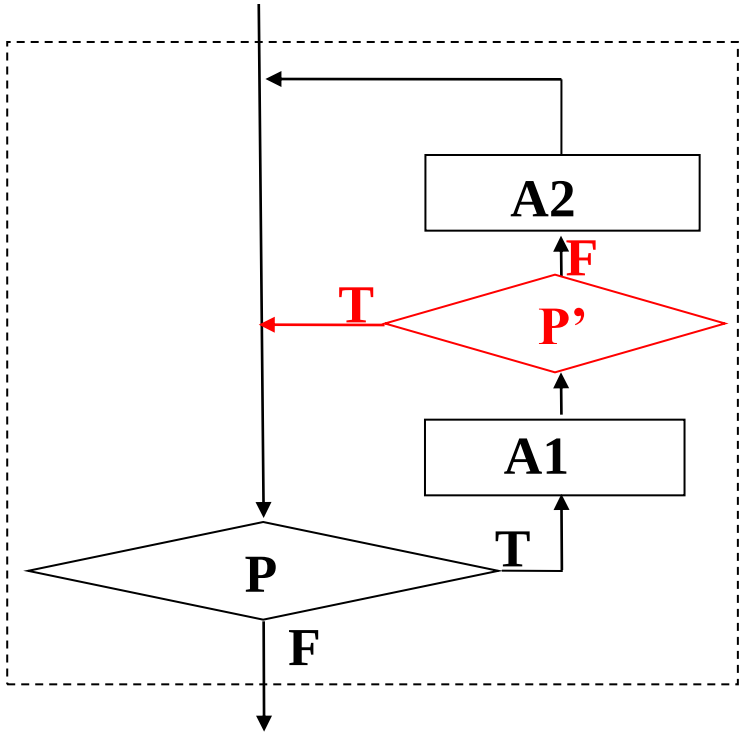
- 如果循环次数不大，改写后效率提高不明显，不必改写，以保持程序简洁。

- **值得提醒的是，对于一些数值型循环计算，改成通项公式计算不一定能得到优化。**
  - 因为通项公式中的运算有可能比原循环体中的运算复杂。
  - 更重要的是，操作数的存储位置会严重影响计算效率。例如，若循环计算涉及的操作数能同时进入缓存，而通项公式中的所有操作数不能同时进入缓存，这种情况下，后者计算效率反而低。
  - 也就是说，程序员应结合循环体内数据的操作与存储情况综合考虑循环流程的优化。

- **顺序流程**
- **分支流程控制方法**
  - 分支流程的基本形式
  - 分支流程的嵌套
  - C 语言其他分支流程控制语句
- **循环流程控制方法**
  - 循环流程的基本形式
  - C 语言其他循环流程控制语句
  - 循环流程的嵌套及其优化
  - **循环流程的折断和接续**
- **综合运用**

- 常规的循环流程可以被**折断**或**接续**，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（折断）或进入下一次循环（接续），从而提高了循环流程的灵活性。



# 循环流程的折断(break)

- C语言中，break语句（通常在循环体中与if结合使用）可以控制循环流程的折断。
  - 执行到break语句，就立即结束循环流程。
  - 例如，输入10个数，依次求和，遇到负数或0就提前终止输入与求和。

```
int d, sum = 0, i = 1;
while(i <= 10)
{
      scanf("%d", &d);
      if (d <= 0) break;
      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(…
```

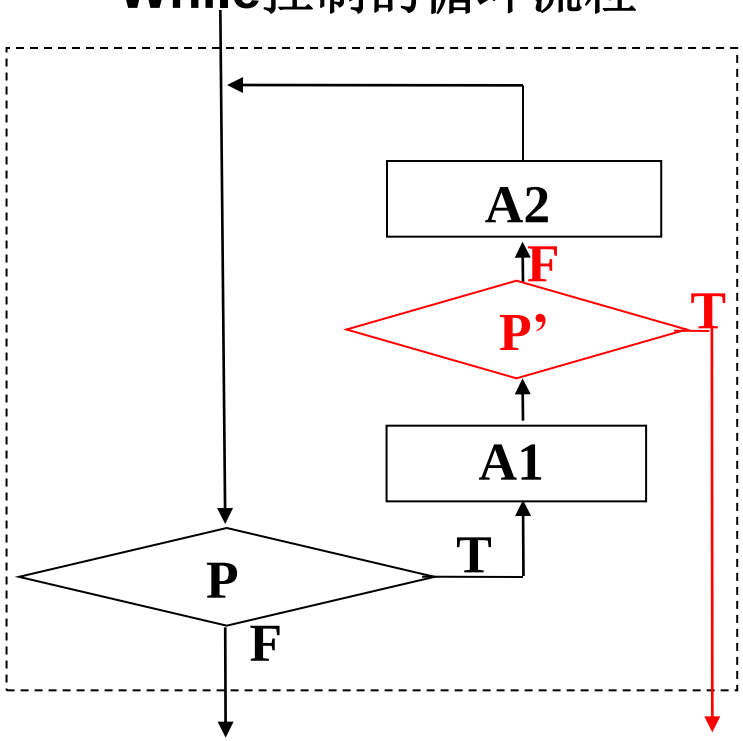
- 折断的等价形式

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

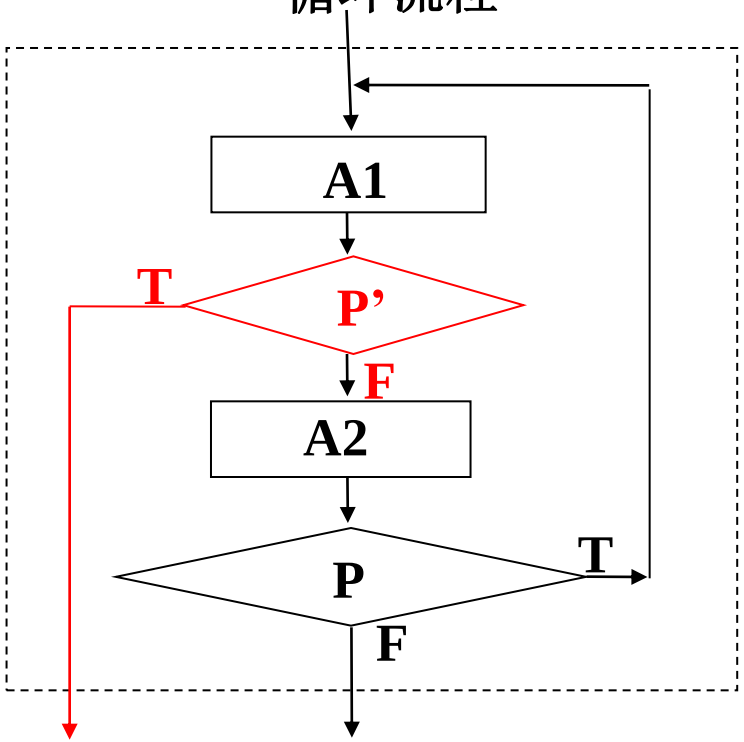
```
int d, sum = 0;
for(int i = 1; i <= 10; ++i)
{
        scanf("%d", &d);
        if(d <= 0) break;
        sum += d;
}▲
printf(…
```

- break辅助的while语句、do-while语句、for语句折断效果是等价的。

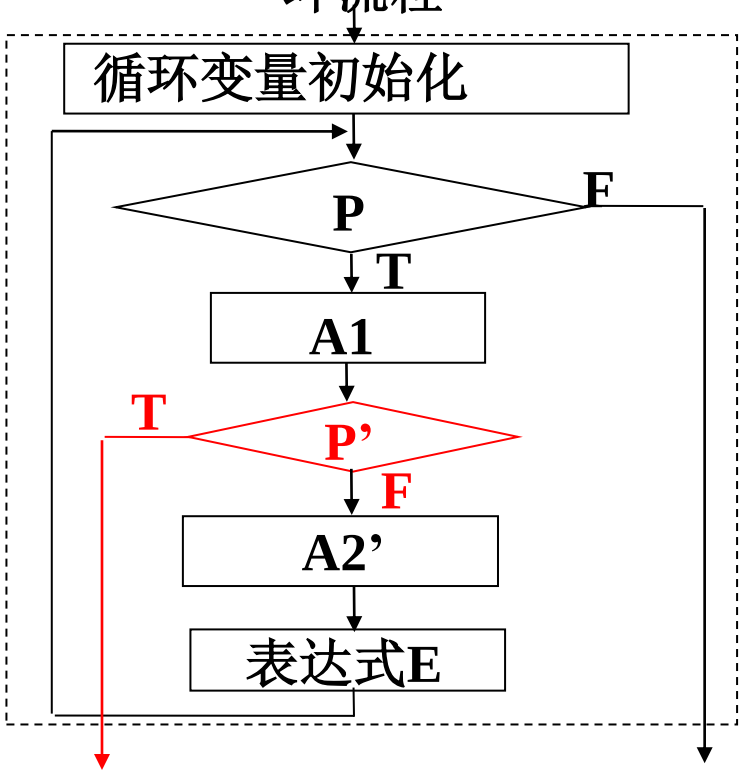
C语言中break辅助While控制的循环流程




C语言中break辅助do-while控制的循环流程




C语言中break辅助for控制的循环流程



## 用break语句协助改写for语句

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

等价于:

```
int sum = 0;
int i = 1;
for( ; ; ) //这种写法虽然编译不出错，但不提倡
{     if(i <= 100)
      {     sum += i;
            ++i;
      }
      else
            break;
}  // ☹
```

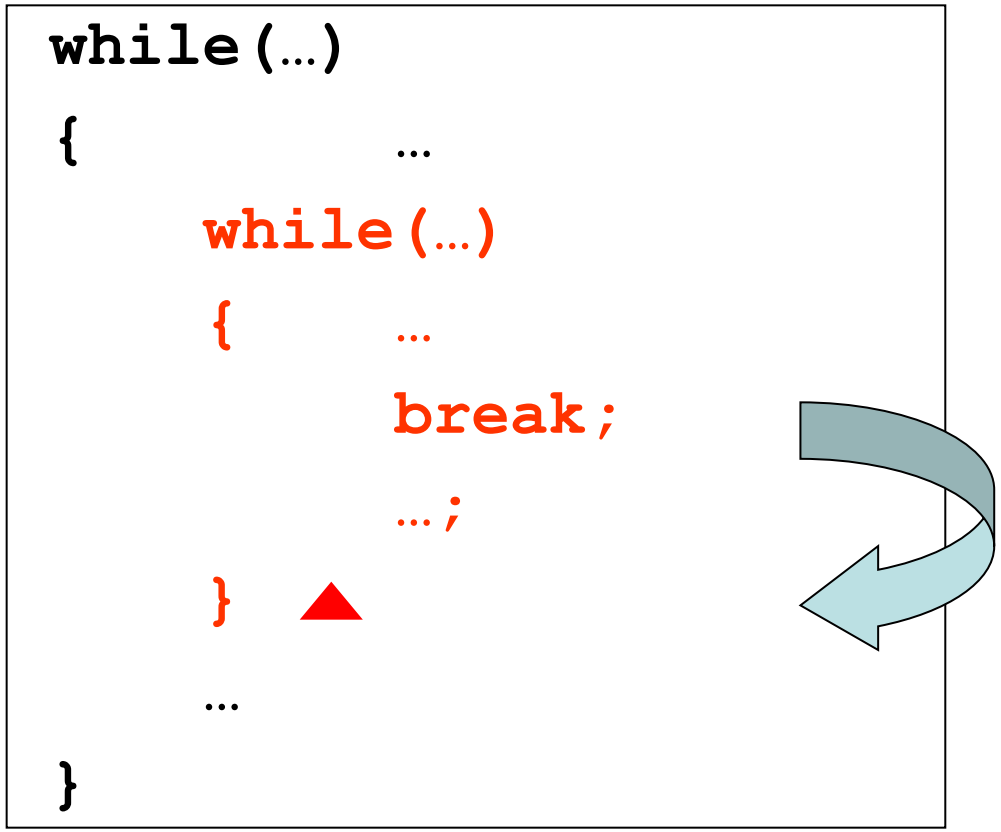
# 用goto语句控制循环流程的折断

```
  int d, sum = 0;
  for(int i = 1; i <= 10; ++i)
  {
     scanf("%d", &d);
     if(d <= 0)
          goto LOOP1;
     sum += d;
  }
LOOP1:
  printf(…
```

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

- 尽量用break语句，而不是goto语句

- 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");
}
```

```
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:  ;
```

- 如果使用goto语句，要注意不能跳过变量的初始化。比如，

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

- 不要用goto语句将流程从一个循环外部转入该循环的内部，以免跳过变量的初始化

```
        …
        goto LOOP3;      // ×
        …
        for(…)
        {
              int y = 10;
LOOP3:              …
        }
        …
```

# 循环流程的接续(continue)

- **C语言中提供了一种continue语句（通常在循环体中与if结合使用），可以控制循环流程的接续。**
  - 执行到continue语句，就跳过循环体后部的任务，流程转向循环的头部（对于while、do-while语句是进行下一次条件判断，对于for语句是计算表达式E）。

- **与break辅助循环流程控制时的区别：**
  - 执行break后，不再进行条件判断，直接结束流程；
  - 执行continue后，接着进行下一次循环.

- **输入10个正数，累计和，如果输入负数，忽略。**

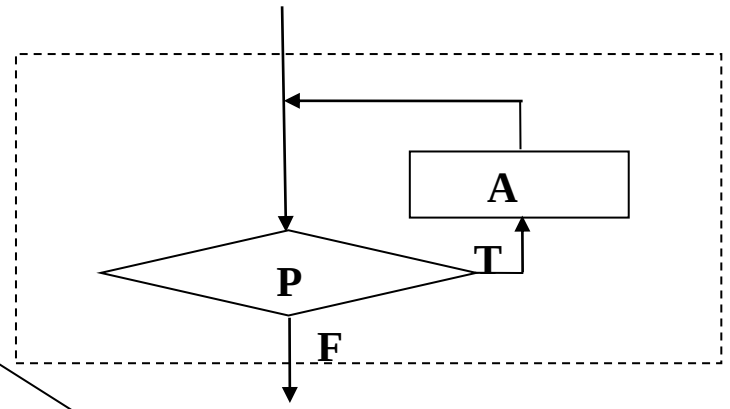
```
int d, sum = 0;
int i = 1;
while(i <= 10)
{
    scanf("%d", &d);
    if (d <= 0) continue;
    sum += d;
    ++i;
}
printf(…
```

等价于：

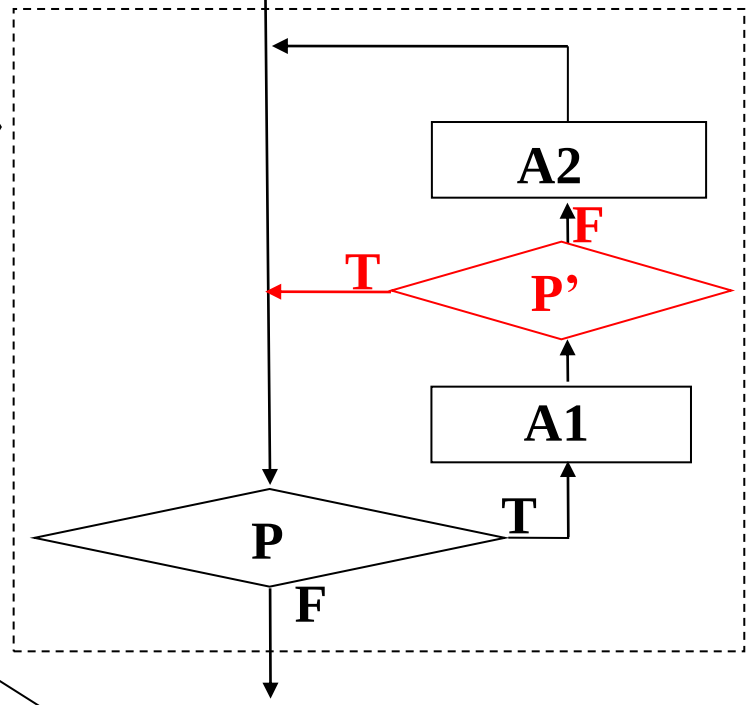
```
1
2
-3
4
5
6
7
8
9
10
11
sum=63
```

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

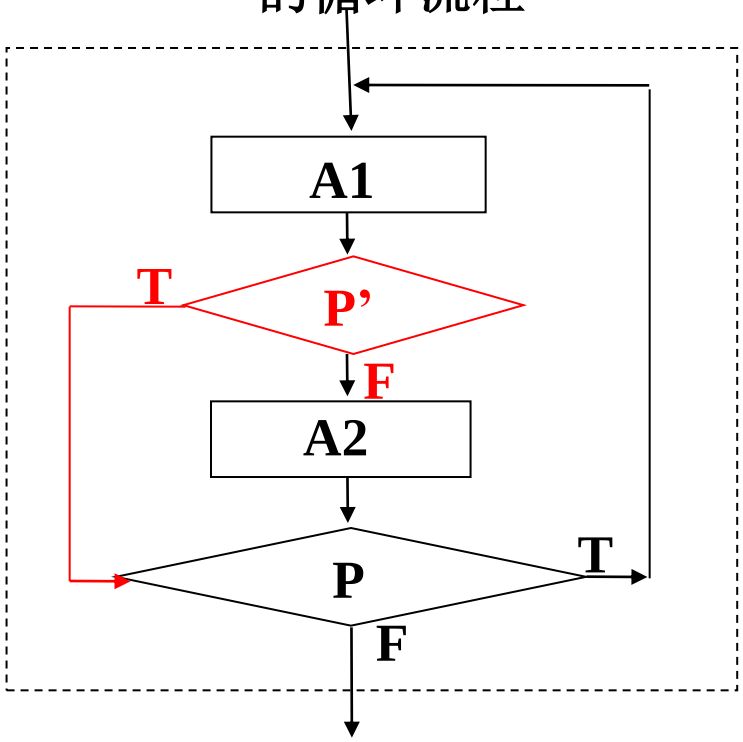
# 接续的等价形式

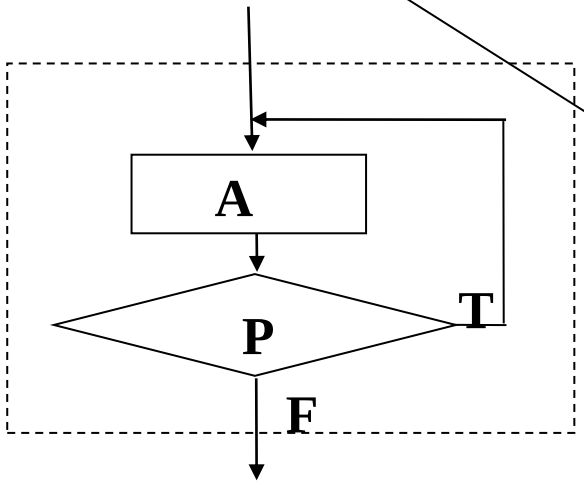



C语言中continue辅助while控制的循环流程




C语言中continue辅助do-while控制的循环流程



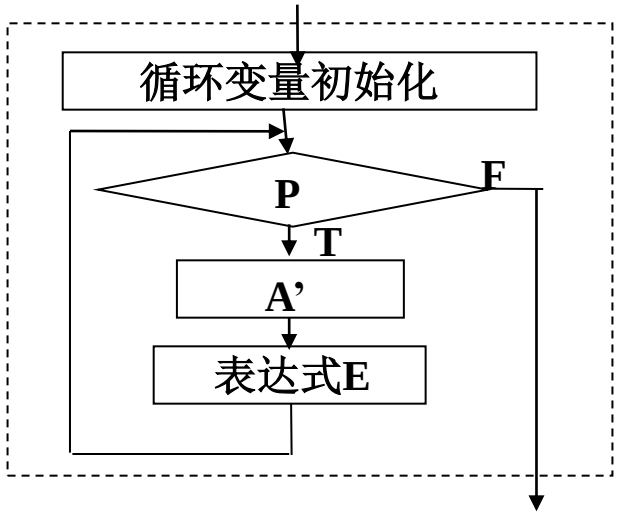



# continue辅助for控制的流程

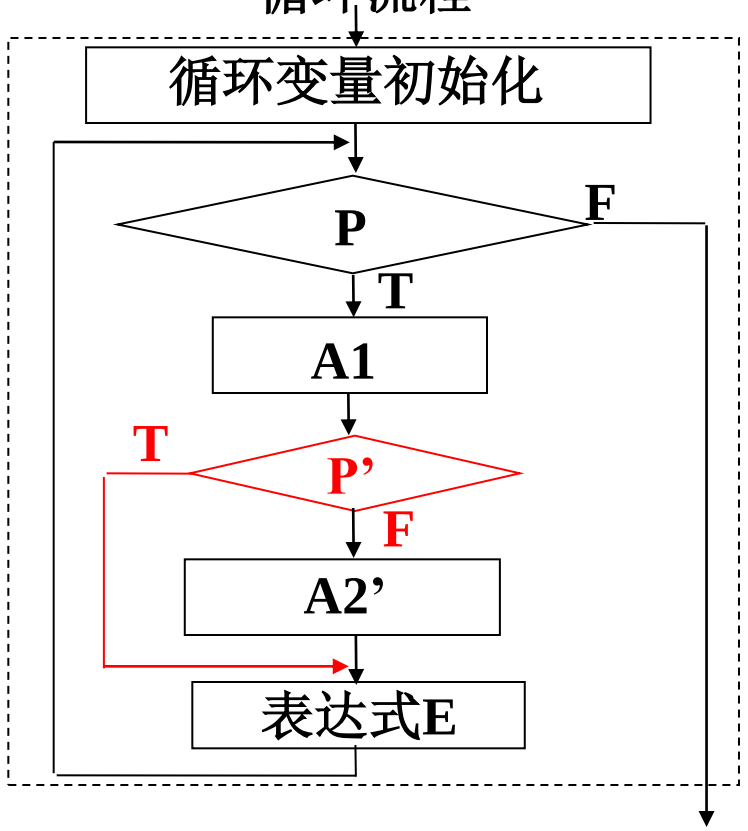
```
int d, sum = 0;
for(int i = 1; i <= 10; ++i)
{
      scanf("%d", &d);
      if(d <= 0) continue;
      sum += d;
}
printf(…;
```

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

- continue辅助for控制的流程 与 continue辅助while/do-while控制的流程不等价




C语言中continue辅助for控制的循环流程



# 除非：

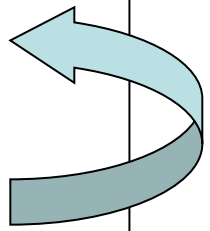
- continue辅助for控制的流程

```
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(…)
        {       …
                continue;
                …;
        }
        …
}
```



# 用goto语句控制循环流程的接续

```
while(...)
{     ......
      ... goto LOOP4;
      ......
 LOOP4:    ;
}
```

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

- 尽量用continue语句，而不是goto语句实现这种循环流程的控制。

# 循环的重要性

- **完成重复操作任务的程序，如100!**
  - 1*2*3*…*100?
    - 编译器往往限制运算符的个数
    - 繁杂
- **完成重复操作任务的程序，如n！(n往往在程序执行期间才能确定)**
  - ?

- **程序设计中常常需要运用循环流程控制方法，甚至需要综合分支流程控制方法，才能实现计算任务。对于一个具体的计算任务，往往要先进行深入仔细的分析，才能发现其中的重复性或带有选择性的重复性计算特征，然后运用恰当的流程控制方法实现其中的流程控制。**

- **顺序流程**
- **分支流程控制方法**
  - 分支流程的基本形式
  - 分支流程的嵌套
  - C语言其他分支流程控制语句
- **循环流程控制方法**
  - 循环流程的基本形式
  - C语言其他循环流程控制语句
  - 循环流程的嵌套及其优化
  - 循环流程的折断和接续
- **综合运用**

# 分类

- 例1.7 设计程序，将用24小时制的时间( 0:00 ~ 23:59)转换为12小时制的时间(12:00am ~ 12:59am, 1:00am ~ 11:59am, 12:00pm ~ 12:59pm, 1:00pm ~ 11:59pm)

```
char cNoon = 'a';
…
if(iHour >= 12)
      cNoon = 'p';
if(iHour == 0)
      iHour = 12;
else if(iHour > 12)
      iHour = iHour - 12;
printf("in 12-hour format : %d:%d", iHour, iMinute);
if(cNoon == 'p')
      printf(" pm \n");
else
```

- 例1.8 设计程序，求所有的十进制三位水仙花数（这种数等于其各位数字的立方和，例如，$153 = 1^3 + 3^3 + 5^3$）。

- [分析] 对于一个十进制三位数，其百位数字可以是1, 2, …, 9，其余两位数字可以是0, 1, 2, …, 9，假设分别用i, j, k表示各位数字，则这个三位数为i*100 + j*10 + k。于是可以通过依次改变i, j, k，列举出所有的三位数，符合条件的数即为所求。
- 这种**依次列举**（又叫枚举）**试探**的过程，即穷举法，是计算机求解问题的常用思路，可以用循环流程实现；判断条件可以用分支流程实现。

# 穷举法

…

```
for(int i = 1; i <= 9; ++i)
{
    for(int j = 0; j <= 9; ++j)
    {
        for(int k = 0; k <= 9; ++k)
            if(i*100 + j*10 + k == i*i*i + j*j*j + k*k*k)
                printf("%d \n", i*100 + j*10 + k);
    }
}
```

…

重复计算

…

```
for(int i = 1; i <= 9; ++i)
{
    int n0 = i * 100, sum0 = i * i * i;
    for(int j = 0; j <= 9; ++j)
    {
        int n1 = n0 + j * 10, sum1 = sum0 + j * j * j;
        for(int k = 0; k <= 9; ++k)
            if(n1 + k == sum1 + k * k * k)
                printf("%d \n", n1 + k);
    }
}
```

…

- 例1.9 （斐波那契Fibonacci数列）有一对兔子，从出生后第3个月起每个月生一对兔子，小兔子长到第3个月后每个月又生一对兔子，假设所有兔子都不死，求第n个月的兔子总数（对）。

**1，1，2，3，5，8，13，…**

```
fib(n) ≠ fib(n-2) + fib(n-1)
```

**Fibonacci 数的定义如下：**

$$fib(n) = \begin{cases} 1 & (n=1) \\ 1 & (n=2) \\ fib(n-2) + fib(n-1) & (n \geq 3) \end{cases}$$

**要想求第n项的值，必须先依次计算前面n-1项的值。**
**这种基于前面的计算结果逐步递推计算的过程叫迭代法，可以运用循环流程来实现。**

# 迭代法

```
…
int main()
{     int n;
      scanf("%d", &n);
      int fib_1 = 1, fib_2 = 1, temp = 2;
      for(int i = 3; i <= n; ++i)
      {     temp = fib_1 + fib_2;
            //计算第i项
            fib_1 = fib_2;
            //第i-1项为下一个i的第i-2项
            fib_2 = temp;
            //第i项为下一个i的第i-1项
      }
      printf("第 %d 个月有 %d 对兔子.\n", n, temp);
      return 0;
}
```

以时间换空间

```
…
int main()
{     int n;
      scanf("%d", &n);
      int fib_1 = 1, fib_2 = 1
      for(int i = 3; i <= n; ++i)
      {     temp = fib_1 + fib_2;
            //计算第i项
            fib_1 = fib_2;
            //第i-1项为下一个i的第i-2项
            fib_2 = temp;
            //第i项为下一个i的第i-1项
      }
      printf("第 %d 个月有 %d 对兔子.\n", n, temp);
      return 0;
}
```

```
fib_2 = fib_1 + fib_2;
//计算第i项，并作为下一个i的第i-1项
fib_1 = fib_2 - fib_1;
//第i-1项作为下一个i的第i-2项
```

fib_2

# 穷举法、分类法

- 例1.10 设计程序，统计输入行（以“#”结尾）中单词的个数。

- **[分析] 通常，输入文本中的单词之间是用空格分隔的，但不排除两个单词之间有多个空格的情况，还有用制表符或回车换行符分隔的情况，所以不能用统计空格数的办法来统计单词个数，应该在每次由分隔符变为单词字符时进行计数。** 设置一个标志用来表征状态的变化

```
if(ch == ' ' || ch == '\t' || ch == '\n')
      flag = 'F';
else                          //不是分隔符，即为单词字符
{     if(flag == 'F')//之前不是单词字符
            ++nWord;
      flag = 'T';
}
```

```
char ch, flag = 'F';
int nWord = 0;
printf("Input line:");
ch = getchar();
while(ch != '#')
{     if(ch == ' ' || ch == '\t' || ch == '\n')
            flag = 'F';
      else                          //不是分隔符，即为单词字符
      {     if(flag == 'F')//之前不是单词字符
                  ++nWord;
            flag = 'T';
      }
      ch = getchar();
}
```

# 穷举法、分类法

- 例1.11 统计交通流量*。

路边设置一车辆探测器，探测器用线路连接到计算机，
当有车辆通过时，探测器传送信号1给计算机，
探测器中有一计时器，每秒钟发出一个数字信号2传给计算机，
该计算机从开始探测时计时，探测结束时传递一个数字信号0给计算机。

**1 2 1 1 2 2 2 1 2 1 2 0**

**编写程序读入这一系列的信号并输出：**
**①进行了多长时间的统计调查；**
**②记录到的车辆数；**
**③在车辆之间最长的时间间隔是多少。**

6秒内，有5辆车通过，
最长隔3秒有车通过

## 分析

输入信号(sign)
- 1(车辆通过)
- 2(秒钟信号)
- 0(探测结束)

输出结果
- ①进行统计的时间(seconds)
- ②记录到的车辆数(nums)
- ③在车辆之间最长的时间间隔(longest)
  由此派生出的车辆间隔变量(inter)

根据以上数据特性，可确定数据类型为整型(int)

# 自顶向下，逐步求精

一级流程

```
begin
    1.数据定义及初始化
    2.读入探测信号sign
    3.while( sign != 0 )
        对sign进行处理
    4.输出结果
end
```

二级流程

```
while( sign != 0 )
{   if ( sign == 1 )
        3.1处理车辆信号;
    else if ( sign == 2 )
        3.2处理计时信号;

    3.3读入下一个sign;
}
```

三级流程

```
begin
    3.1.1 车辆计数nums++;
    3.1.2 处理最大间隔
      if(longest<inter)
                longest=inter;
    3.1.3 为下一个间隔计数作准备
       inter=0;
end
```

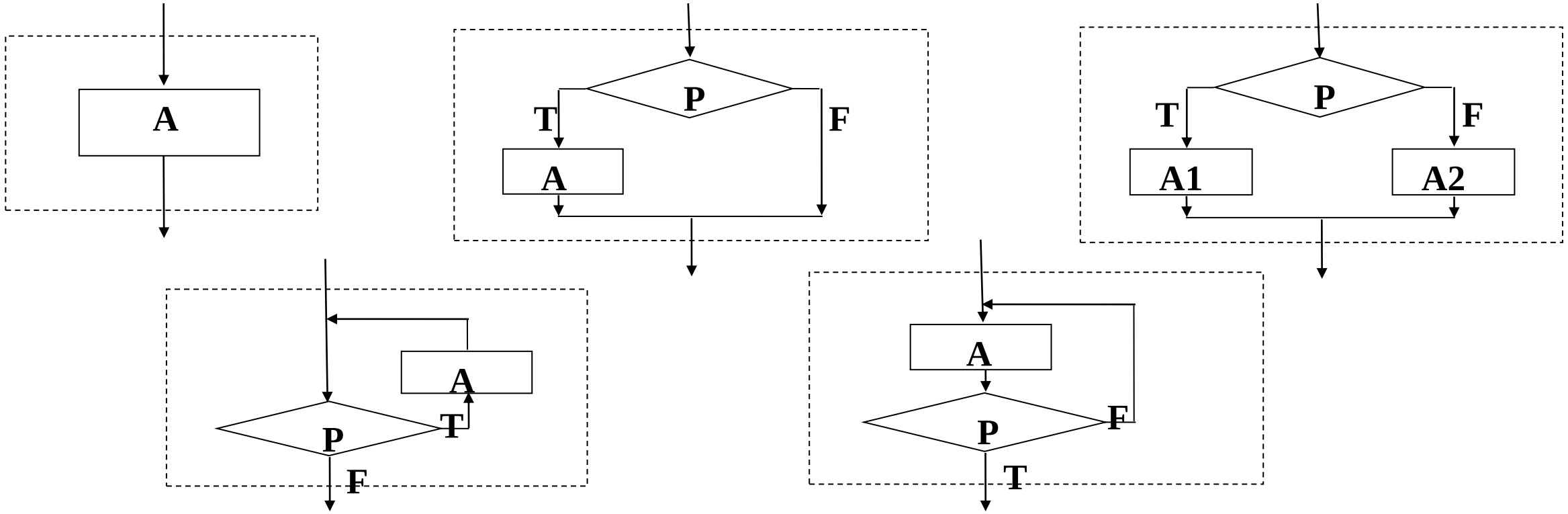
```
begin
    3.2.1 总时间累加器seconds++
    3.2.2 间隔计数器inter++;
end
```

```
…
    int sign;
    int nums=0, seconds=0, inter=0, longest=0;
    scanf("%d", &sign);
    while(sign != 0)
    {    if(sign == 1)
         {    nums++;
              if(inter > longest)
                   longest = inter;
              inter = 0;
         }
         else if(sign == 2)
         {    ++seconds;
              ++inter;
         }
         scanf("%d", &sign);
    }
    printf("seconds = %d, nums =%d, longest=%d \n", seconds, nums, longest);
    …
```

**“假定最大值”longest应初始化成一个尽量小的数，之后根据进一步所获信息，通过赋值操作逐步修正。**

# 基本流程

- **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；
  - 流程内不存在“死循环”。



# C语言的流程控制语句

- **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）
- 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
  - return（参见ch2）
- **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**

# 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，都可嵌套，它们之间可以互相替代。**
- **对于某个具体的问题，用其中的某个语句来描述可能会显得比较自然和方便，使用三种语句的一般原则：**
  - 计数控制的循环，用for语句
  - 事件控制的循环，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。
- **由于for语句的结构性较好，循环流程的控制均在循环顶部统一表示，更直观，即for语句可显式地表示出：循环变量初始化、循环结束条件以及下一次循环准备，很多情况下都采用for语句。**

- **辅助控制语句会提高流程控制的灵活性，但是会降低程序的可读性和可靠性；**
- **流程清晰的程序，程序中的每一个流程单元都应是单入口/单出口，辅助控制语句（特别是goto语句）会破坏这个规则，所以，不提倡使用goto语句；**
- **理论上可以证明，所有的程序都可以不用goto语句来实现：**
  - 往回的转移（backward）可用循环控制语句实现
  - 向前的转移（forward）可用分支控制语句实现

# 小结

- 分支流程及其控制方法
  - **if... / if...else...**
  - **switch... (break)**
  - **嵌套**
- 循环流程及其控制方法
  - **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≈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
- 作业：见课程网站

# Thanks !