

回顾：一个完整的例子（C++）

- ▶ 例0 计算一组圆（直径为n以内的正整数）的周长之和（计量单位为米）。

```
#include <iostream>
using namespace std;
const double PI = 3.14;
```

```
int main( )
{
    int n, d = 1;
    double sum = 0;
    char ch = 'm';
    cout << "Input n: " ;
    cin >> n;
    .....
    return 0;
}
```

while(d <= n)

```
{
    sum = sum + PI * d;
    d = d + 1;
}
```



```
cout << "The sum is: " << sum;
cout << ch;
```

跑圈



一周的学习和生活...

从周一到周五 ...

起床



吃早餐



上课



吃午餐



午休



上课



锻炼



晚自习

按天重复... 循环!



3.3 循环

郭延文

2019级计算机科学与技术系

循环流程控制方法

- ▶ 循环流程的基本形式
- ▶ 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
- ▶ 循环流程控制方法的综合运用



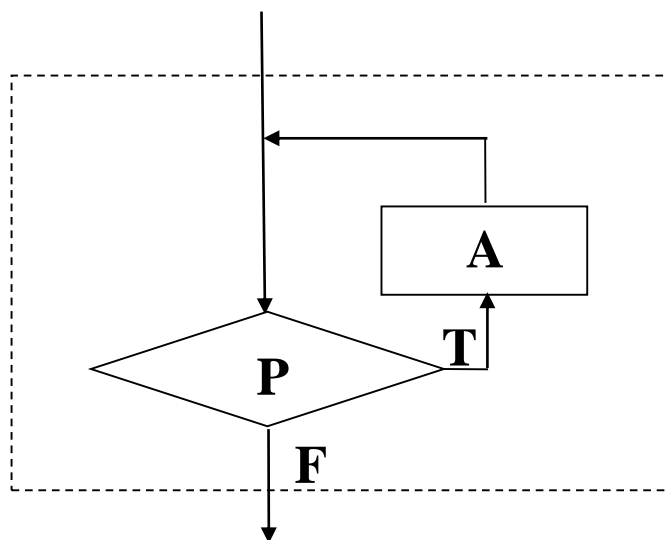
循环流程

- ▶ 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，即循环体的“状态”可能不同）
- ▶ 循环流程用于这种“重复性”计算场合，是程序设计的一种重要流程
- ▶ 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



while语句

- ▶ 实现循环流程控制的一种语句，其格式为：

```
while (<条件P>)
```

```
{
```

```
    <任务A>
```

```
}
```

- ▶ 条件P一般是带有关系或逻辑操作的表达式
- ▶ 任务A是while语句的子句，通常是一个复合语句，写在下一行，并缩进
- ▶ 条件P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，语句结束

例: 求N（比如100）以内自然数的和

```
#include <stdio.h>
#define N 100
int main( )
{
    int i = 1, sum = 0;
    while(i <= N)                //该行没有分号
    {
        sum = sum + i;           //等价于 sum += i;
        i = i + 1;               //等价于 i++;
    }
    printf("Sum of integers from 1 to %d: %d \n", N, sum);
    return 0;
}
```

类似地: **sum -= i** 等价于 $\text{sum} = \text{sum} - i$;
i-- 等价于 $i = i - 1$

例2.1 求N（比如100）以内自然数的和

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

```
#define N 100
```

```
int main( )
```

```
{
```

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

```
    while(i <= N)
```

```
    {
```

```
        sum = sum + i; //可改为sum += i;
```

```
        i = i + 1;
```

```
        //改为i++;
```

```
    }
```

```
    printf("Sum of integers from 1 to %d: %d\n", N, sum);
```

```
    return 0;
```

```
}
```

类似地: $i =$

$i=1, \text{ sum}=0$

$i=i+1;$

$\text{sum}=\text{sum}+i;$

$i \leq N?$

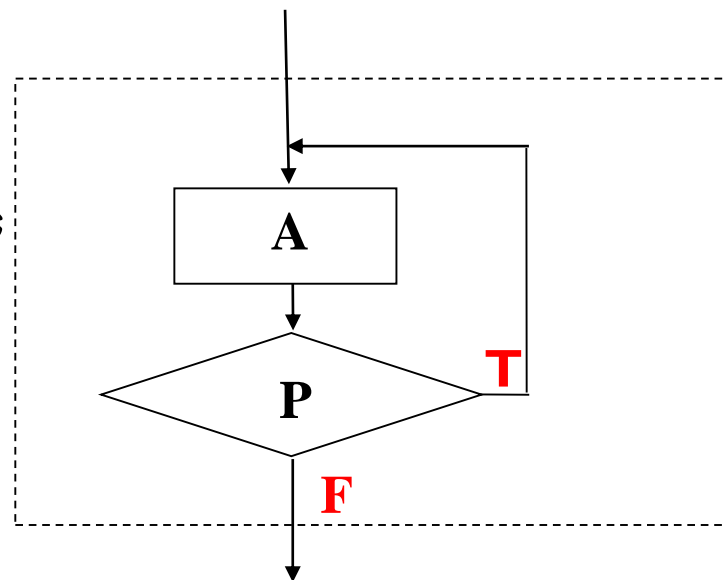
T

F

输出sum

循环的另一种形式

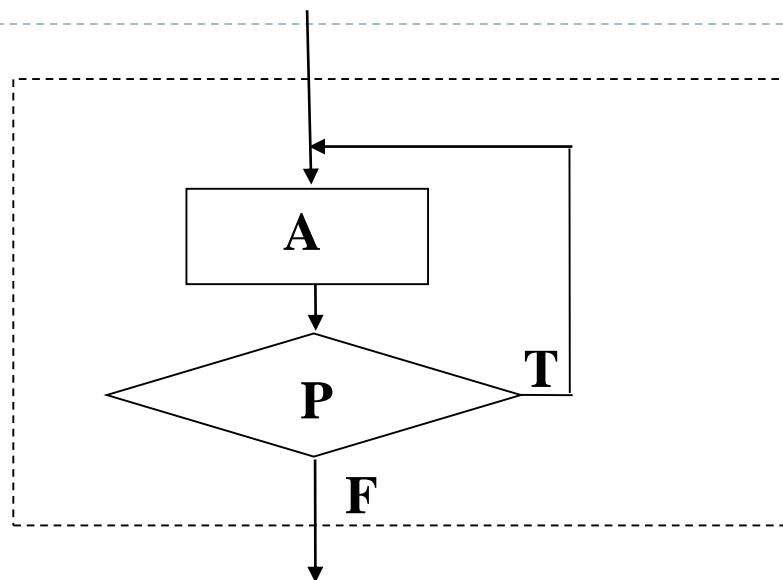
- ▶ 包含一个任务和一个条件判断
 - ▶ 先执行一次任务A，再判断条件P
 - ▶ 当条件P成立时继续执行任务A，并再次判断条件P，如此循环往复；
 - ▶ 当条件P不成立时，该流程结束



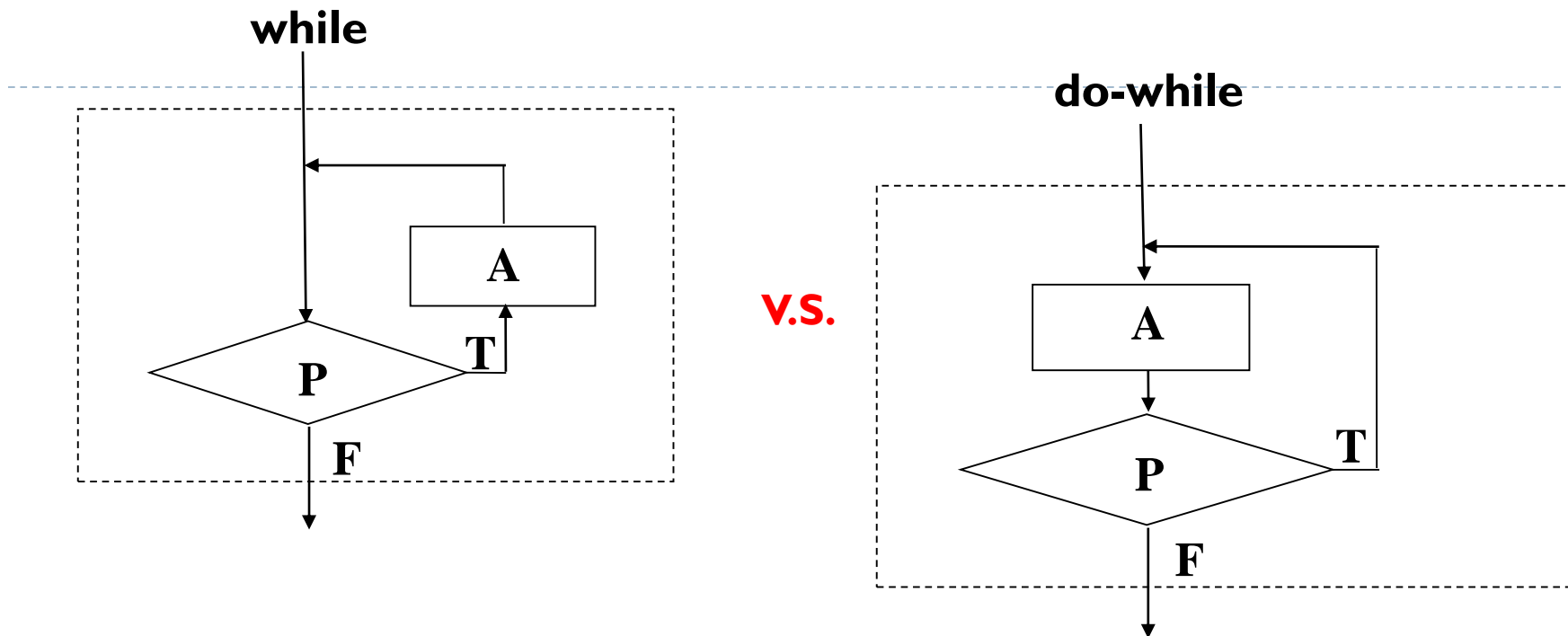
do while语句

“无论如何先干一票”！

```
do  
{  
    <任务A>  
} while (<条件P>);
```



- ▶ 任务A（首先执行）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，再次执行任务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



- ▶ **while** 语句：条件P先判断一次，如成立，在执行A之后继续判断下一次，而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（条件P一开始就不成立）

循环语句要避免“死”循环！

“唉！程序为何执行不完了，半天没动静？”

例: 求N（比如100）以内自然数的和

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

```
#define N 100
```

```
int main( )
```

```
{
```

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

```
    do
```

```
    {
```

```
        sum += i;
```

```
        i ++;
```

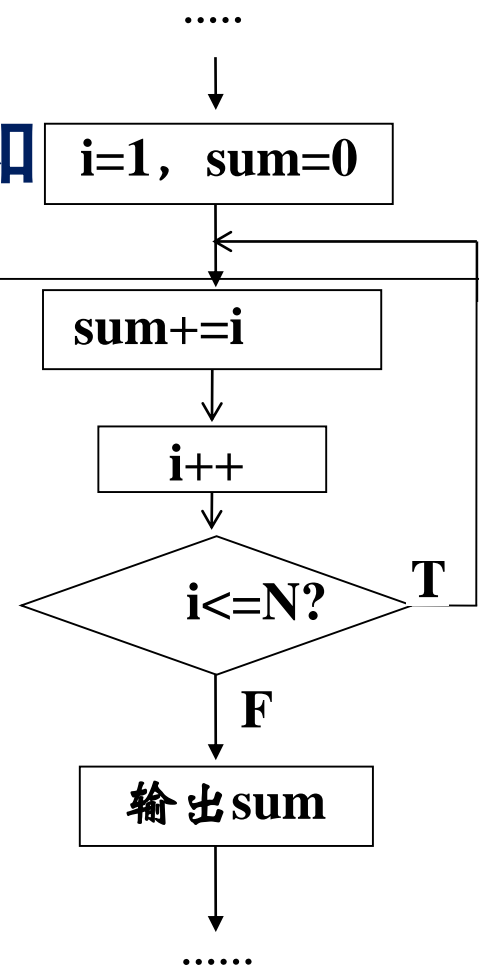
```
    } while(i <= N);
```

//该行有分号

```
    printf("Sum. of integers from 1 to %d: %d \n", N, sum);
```

```
    return 0;
```

```
}
```



//思考： 若初值 $i=101$ ， 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。

while / do while 书写注意:

▶ 多写或少写分号

```
i = 0;  
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，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

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

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

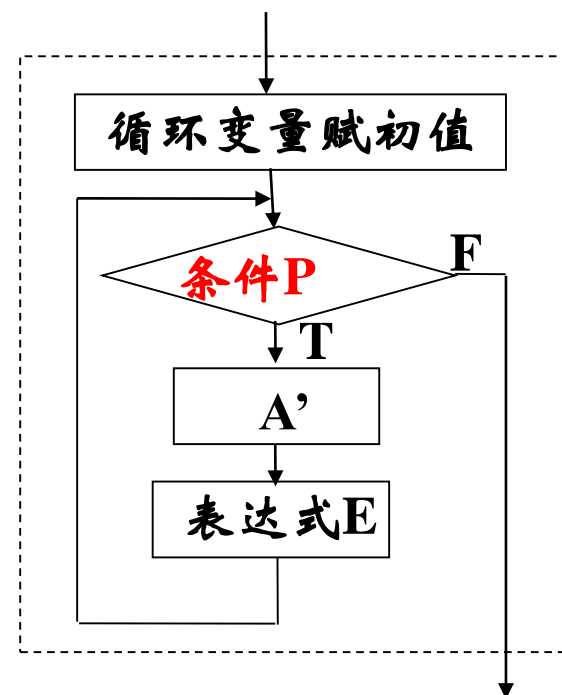

for 语句

▶ 语法形式:

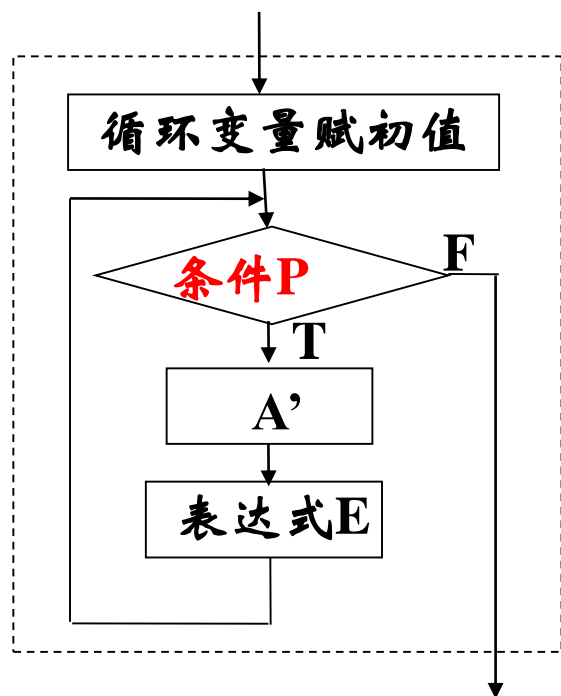
```
for (<循环变量赋初值>; <条件P>; <表达式E>)  
{  
    <任务A'>  
}
```

▶ 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',
- ▶ 计算表达式E
- ▶ 判断条件P, 如此循环往复; 当条件P不成立时, 结束该流程



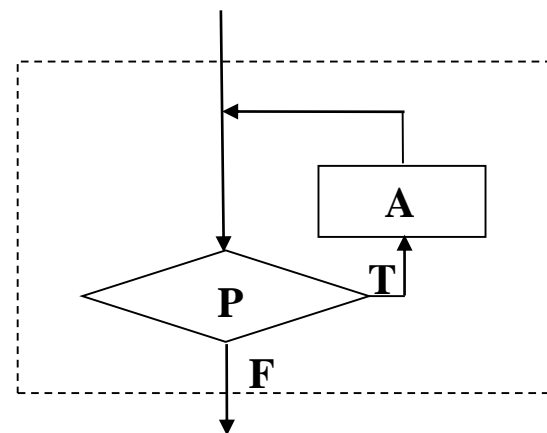
```
int i, sum=0;
for (i=1; i<=100; i++)
    sum += i;
```



- for语句一般将循环变量放入循环语句内赋初值；表达式E一般包含对循环变量的操作，往往称作步长，循环变量按步长增大或减小，导致循环结束

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

V.S.



while或do...while语句的循环变量通常在循环语句前赋初值，在循环体内改变值

书写注意!

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

空语句

导致sum为101

循环体用**括号**括起来！

- ▶ 如果循环体含有多条语句，则一定要用花括号将多条语句括起来；否则，只有第一条语句能被循环执行，从而导致执行结果错误

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

-
- 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，）

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

或：

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

- ▶ for语句圆括号里可以定义循环变量，并且该循环变量往往只在for语句内有效，比如，

```
...  
int sum = 0;  
for(int i = 1; i <= N; i++) // i 局部变量  
{  
    sum += i;  
    ...  
}  
i = 3; // 编译出错  
...
```

- ▶ 注：少数编译器（如VC++6.0）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；但不建议再用VC6.0!

goto语句（后面会再详细介绍该语句）

- ▶ goto语句与if语句以及“语句标号”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；不建议用goto语句！

循环流程的嵌套及其优化

- ▶ 循环流程也可以嵌套，即循环体中又含有循环流程
- ▶ 例如：从第1个月到第12个月

```
{  
    从第1周到第4周  
    {  
        从礼拜1到礼拜5  
        {  
        }  
        从礼拜6到礼拜7  
        {  
        }  
    }  
}
```


例：求输入的一个正整数的阶乘

```
#include <stdio.h>
int main()
{
    int n, i = 2, f = 1;    //f存放计算结果，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：每输入一个正整数，计算其阶乘，直到输入0为止

```
#include <stdio.h>
int main( )
{
    int n, i, f;
    printf("Please input an integer (input 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 (input 0 to exit): \n");
        scanf("%d", &n);
    }
    return 0;
}
```

再回顾这个例子

例：每输入一个正整数，计算其阶乘，直到输入0为止

- ▶ 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)           // 有效性判断
{
    // i = 2, f = 1; // 双层循环注意某些变量初值的“还原”
    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，缩进用Tab键

用for或do while实现求阶乘的例子

```
for (; n != 0;) // 有效性判断
{
    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
    .....
}
```

- ▶ **while/do-while** 更适合: 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
- ▶ **for** 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

例2.4 输出一个九九乘法表

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

```
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;
```

\t 制表符
C标准中空白字符有：空
格（' '）、
换页（'\f'）、
换行（'\n'）、
回车（'\r'）、
水平制表符（'\t'）、垂
直制表符（'\v'）
通过实验体会其作用！

}

- ▶ 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。

用while/do while改写

```
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);
```

新要求：只输出乘法表的下三角

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

新要求：只输出乘法表的下三角

.....

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

只输出乘法表的上三角

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

只输出乘法表的上三角

```
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");
```

```
}
```

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

```
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");
```

```
}
```

循环嵌套的优化

- ▶ 嵌套循环中，如果有可能，应将长循环放在最内层，短循环放在最外层，以减少CPU跨切循环层的次数，提高程序的运行效率：

```
for(row = 0; row < 100; row++)           //长循环在最外层，效率低
    for(col = 0; col < 5; col++)
        sum += row * col;
```

- ▶ 可以改写为：

```
for(col = 0; col < 5; col++)             //长循环在最内层，效率高
    for(row = 0; row < 100; row++)
        sum += row * col;
```

循环流程里嵌套分支流程

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

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。

//if...else...是一条语句，这对花括号不加也可以，加上更清晰

- ▶ 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;
```

可以用OpenMP优化！自己网络查阅相关资料

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
- ▶ 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语句可显式地给出：循环变量初始化 + 循环结束条件 + 下一次循环准备)
- ▶ 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

具体问题具体分析！

例：计算从键盘输入的一系列整数的和，要求首先输入整数的个数。（计数控制的循环）



例：计算从键盘输入的一系列整数的和，要求首先输入整数的个数。（计数控制的循环）

```
#include <iostream>
using namespace std;
int main()
{   int n;
    cout << "请输入整数的个数: ";
    cin >> n;
    cout << "请输入" << n << "个整数: ";
    int sum=0;
    for (int i=1; i<=n; i++)
    {   int a;
        cin >> a;
        sum += a;
    }
    cout << "输入的" << n << "个整数的和是: " << sum << endl;
    return 0;
}
```

例：计算从键盘输入的一系列整数的和，要求输入以0结束。
(事件控制的循环)



例：计算从键盘输入的一系列整数的和，要求输入以0结束。
(事件控制的循环)

```
#include <iostream>
using namespace std;
int main()
{   int a,sum=0;
    cout << "请输入若干个整数（以0结束）：";
    cin >> a;
    while (a != 0)
    {   sum += a;
        cin >> a;
    }
    cout << "输入的整数的和是：" << sum << endl;
    return 0;
}
```



例：计算从键盘输入的n个正整数的和，当输入负数或0的时候直接Pass，直到输入n个为止，开始计算和。

`while (a != 0)`

`while (i <= n)`




例：求第n个费波那契(Fibonacci)数
// 1,1,2,3,5,8,13,... (兔生兔问题)



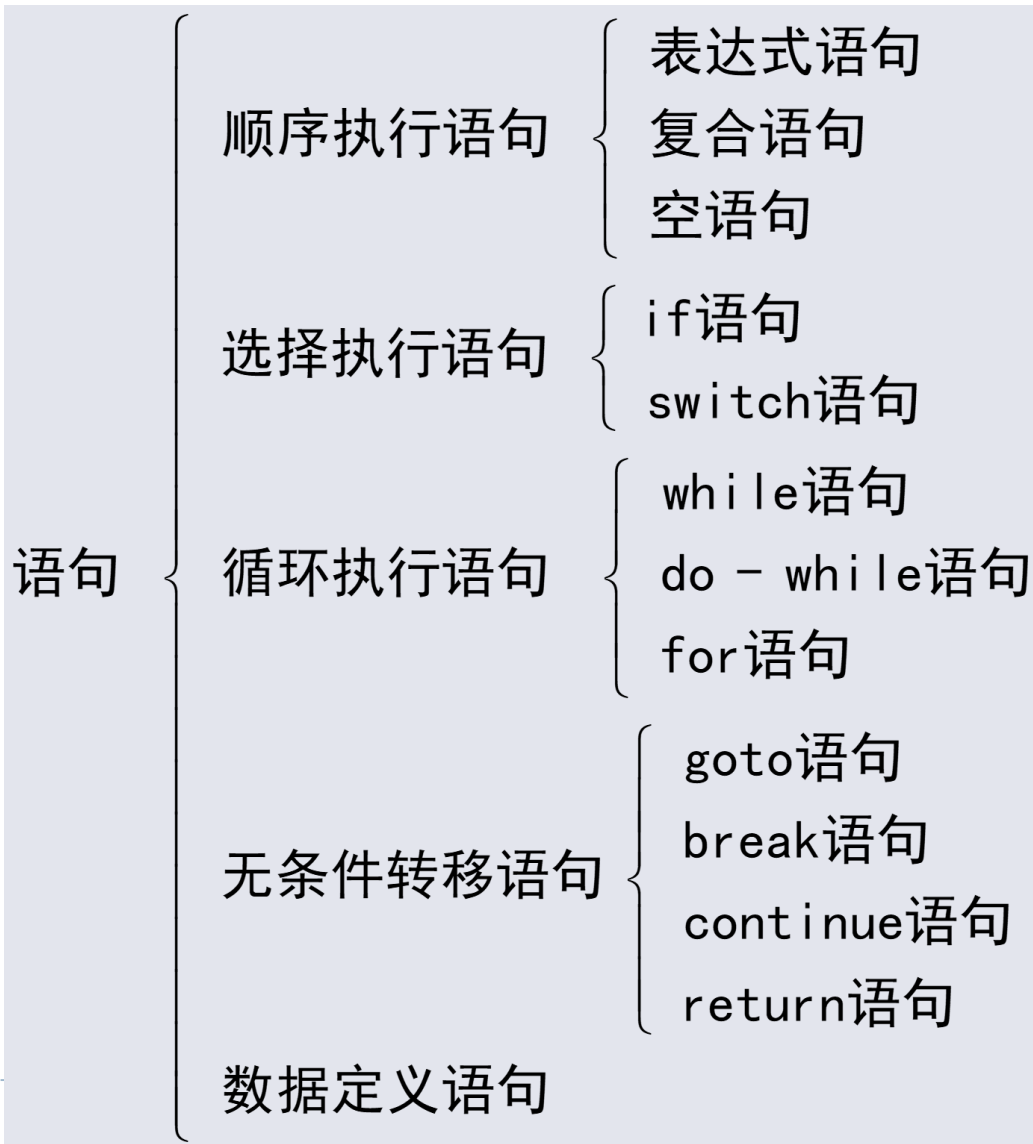
例：求第n个费波那契(Fibonacci)数

// 1,1,2,3,5,8,13,... (兔生兔问题)

```
#include <iostream>
using namespace std;
int main()
{
    int n;
    cin >> n;
    int fib_1=1; //第一个Fibonacci数
    int fib_2=1; //第二个Fibonacci数
    for (int i=3; i<=n; i++)
    {
        fib_2 = fib_1 + fib_2; //计算和记住新的Fibonacci数
        fib_1 = fib_2 - fib_1; //记住前一个Fibonacci数
    }
    cout << "第" << n << "个费波那契数是：" << fib_2 << endl;
    return 0;
}
```



C/C++语句的分类



Q & A



switch语句

- ▶ 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。

```
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
```


while语句

- ▶ 实现循环流程控制的一种语句，其格式为：

```
while (<条件P>)
```

```
{
```

```
    <任务A>
```

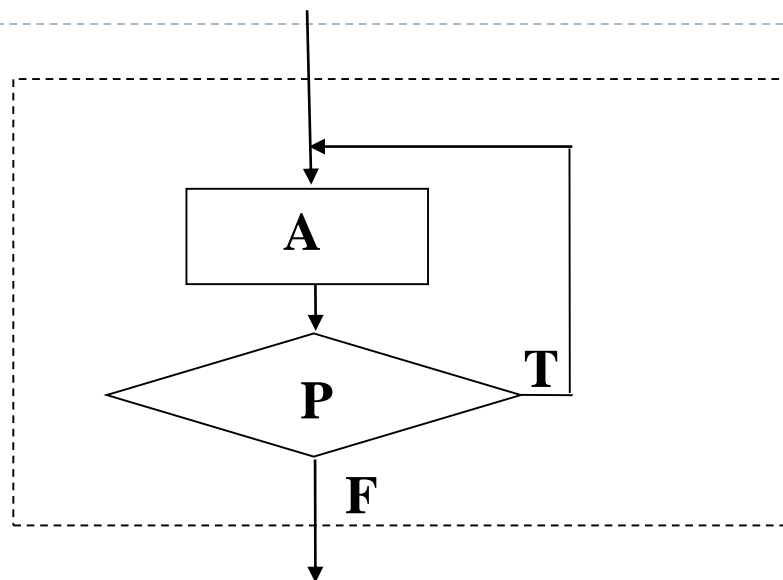
```
}
```

- ▶ 条件P一般是带有关系或逻辑操作的表达式
- ▶ 任务A是while语句的子句，通常是一个复合语句，写在下一行，并缩进
- ▶ 条件P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，语句结束

“无论如何先干一票”！

do while语句

```
do  
{  
    <任务A>  
} while (<条件P>);
```



- ▶ 任务A（首先执行）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，再次执行任务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

for 语句

▶ 语法形式:

```
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',
- ▶ 计算表达式E
- ▶ 判断条件P, 如此循环往复; 当条件P不成立时, 结束该流程

