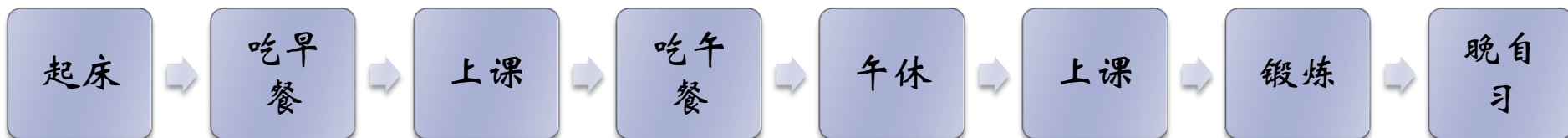


# 一周的学习和生活...

从周一到周五 ...



```
for (i=Monday; i<= Friday; i++) // 举例
```

```
{
```

```
    if (国庆节)
```

```
    {
```



```
        break;
```

```
    }
```

```
    每天的上课、学习生活;
```

```
}  
▶
```

# 3.4 无条件转移控制

郭延文

2019级计算机科学与技术系

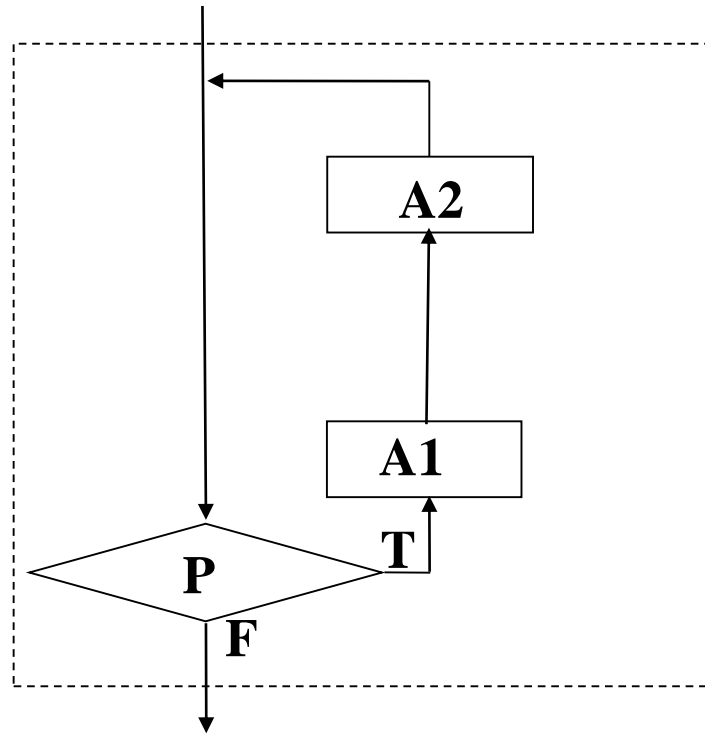
# 主要内容

---

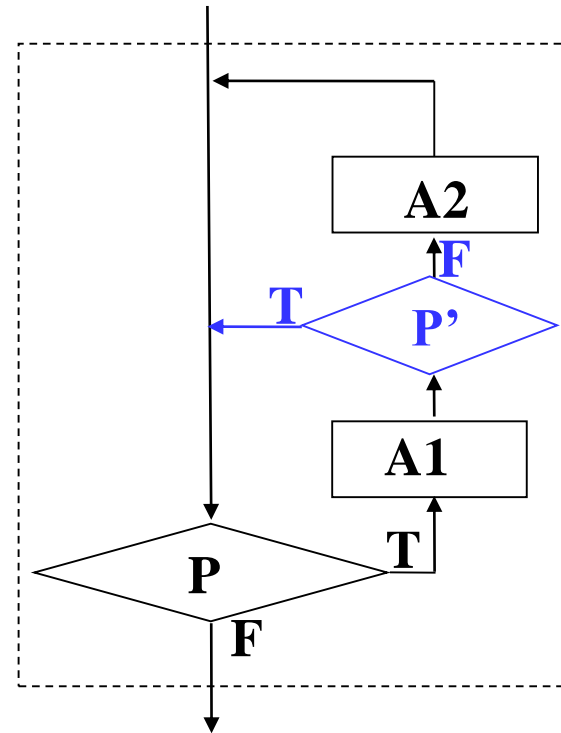
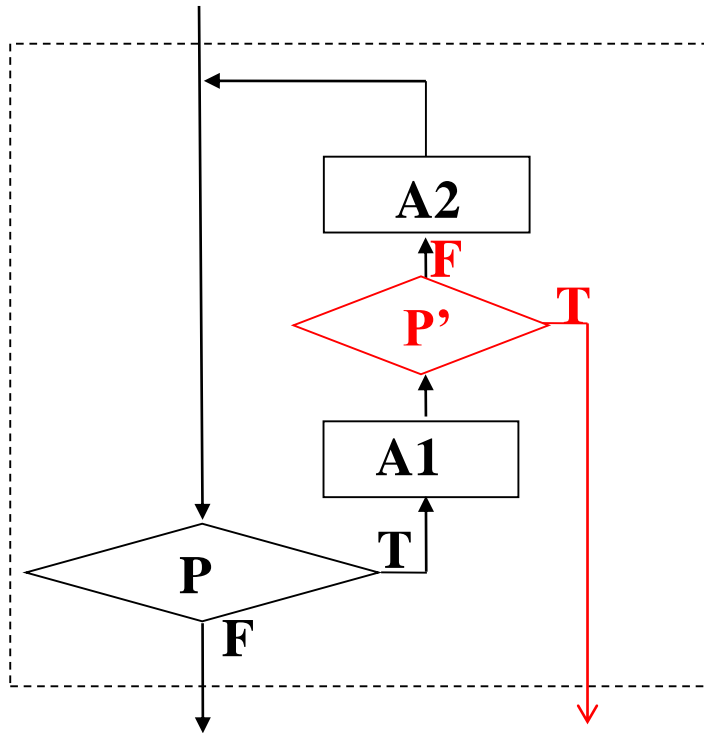
- ▶ 循环流程的基本形式
- ▶ C语言循环流程控制语句
- ▶ 循环流程的嵌套及其优化
- ▶ 循环流程的折断和接续
- ▶ 循环流程控制方法的综合运用



- 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，在执行其中一部分操作后，根据一定情况在相应的语句控制下，**结束整个循环（折断）**或**“提前”进入下一次循环（接续）**，从而提高循环流程的灵活性。



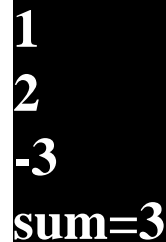
- 常规的循环流程可以被折断或接续，即循环操作往往被分成两部分，在执行其中一部分操作后，根据一定情况在相应的语句控制下，**结束整个循环（折断）**或**“提前”进入下一次循环（接续）**，从而提高循环流程的灵活性。



# 循环流程的折断(break)

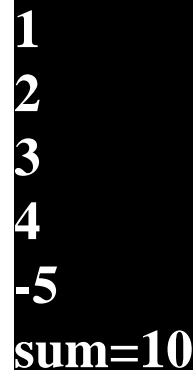
- ▶ C语言中，break语句（通常在循环体中与if结合使用）可以控制循环流程的折断。
- ▶ 执行到break语句，就立即结束循环流程。
- ▶ 例如：输入10个数，依次求和，遇到输入的负数或0就提前终止求和的过程。

```
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

循环只执行了2次.



1  
2  
3  
4  
-5  
sum=10

循环执行了4次.

# 用for/do while改写

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

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

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

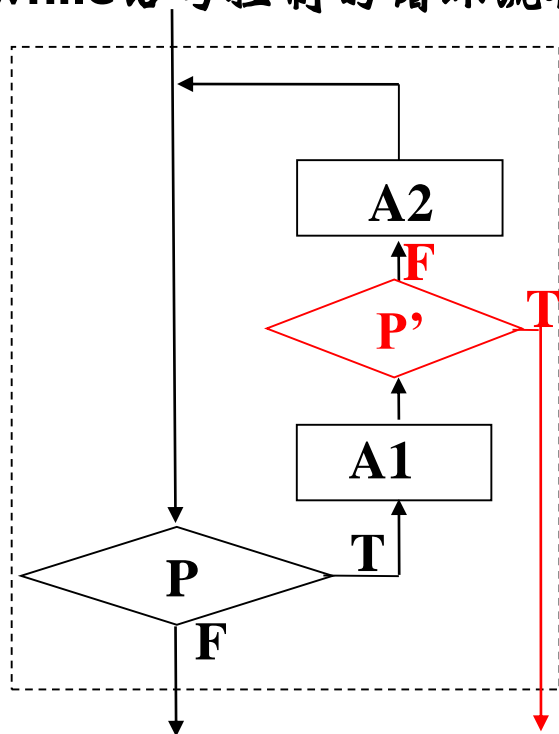
```
int d, sum = 0;
for(int i = 1; i <= 10; i++)
{
    scanf("%d", &d);
    if (d <= 0)
        break;
    sum += d;
}
printf("sum: %d \n", sum);
```

# break “运行流程”

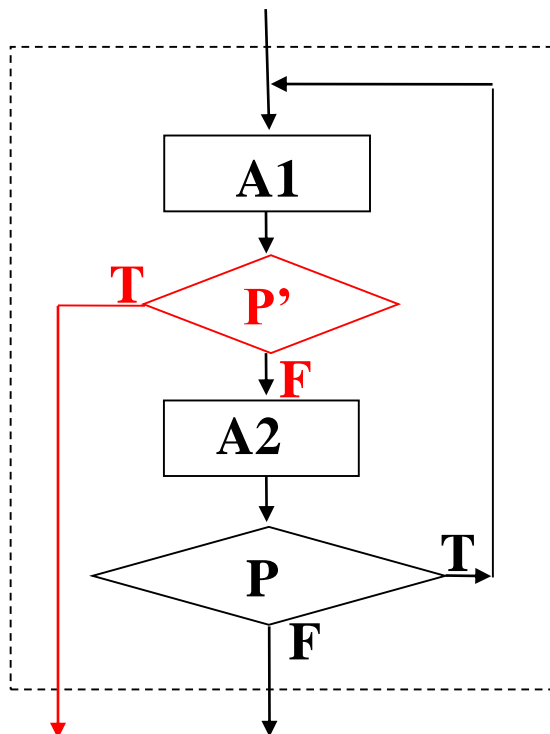
- break辅助的while语句、do-while语句、for语句，当初始循环条件满足时折断效果是等价的。

break辅助

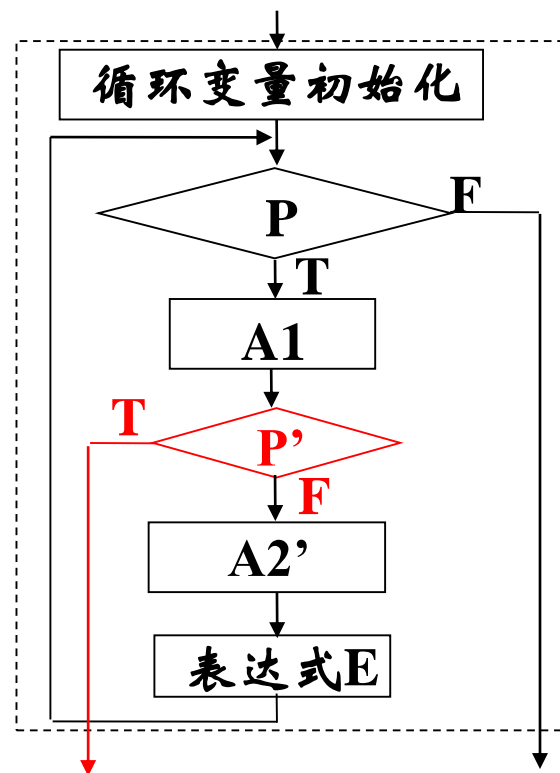
while语句控制的循环流程



break辅助do-while语句



break辅助for语句





# 嵌套循环的break

- ▶ 在嵌套循环中，内层循环体里的break折断内层循环流程。

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



## 例：阅读程序并写出执行结果

```
for(int i = 1; i <= 9; i++)
```

// 外层循环仍然执行9次

```
{
```

```
    for(int j = 1; j <= 9; j++)
```

// 只不过部分内层循环并非每次都  
执行9次

```
{
```

```
    if(i * j > 10)
```

```
        break;
```

```
    printf("%d \t", i * j);
```

```
}
```

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								

## 例：阅读程序并写出执行结果

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

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

# goto语句 - 无条件转移控制语句

---

- ▶ goto语句的格式如下：

goto <语句标号>; // (“代号”)

// <语句标号>为标识符，其定义格式为：

<语句标号>: <语句>



- ▶ goto的含义是：程序转移到特定<语句标号>的语句



# 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("sum: %d \n", sum);
```

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

- ▶ 尽量用break语句，而不是goto语句

## goto语句一个用途：跳出多重循环

---

```
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语句注意

---

- ▶ 不能用goto语句从一个函数外部转入该函数的内部  
(函数体)
- ▶ 不能用goto语句从一个函数的内部转到该函数的外部
- ▶ 允许用goto语句从内层复合语句转到外层复合语句或从外层复合语句转入内层复合语句
- ▶ goto语句不能掠过带有初始化的变量定义



# goto语句使用注意

---

- ▶ goto语句不能跳过变量的初始化！ 例如：

...

while(...)

{

...

if(...)

goto L2;      //错误，因为不能跳过变量y的初始化

...

}

int y = 10;

L2: ...





# goto语句使用注意

---

```
void f()
{   .....
    goto L1;           // 错误
    .....
    while (...)
    {   int x=0;
        L1: ...
        .....
        goto L2;       // 错误
        .....
    }
    .....
    int y=10;
    L2: .....
    .....
}
```

---



# 关于goto语句

---

- ▶ 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
- ▶ 比较合适的应用场合：跳出多重循环！

# continue语句：循环流程的接续

---

- ▶ 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，提前进入下一次循环
- ▶ 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

## 例：输入10个正数，求和；如果输入负数，则忽略该负数

等价于：

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

```
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("sum: %d \n", sum);
```

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：输入10个数，求其中的正整数之和。

---

**// 如何改写这个程序？**

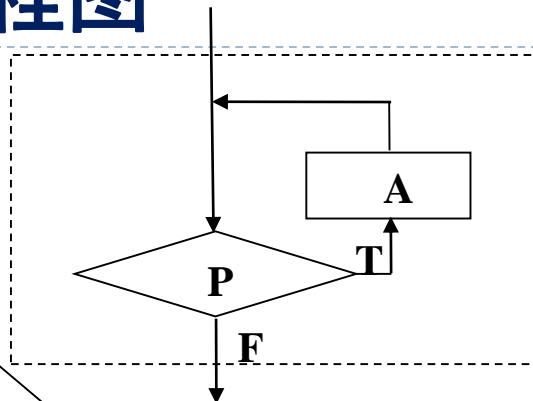
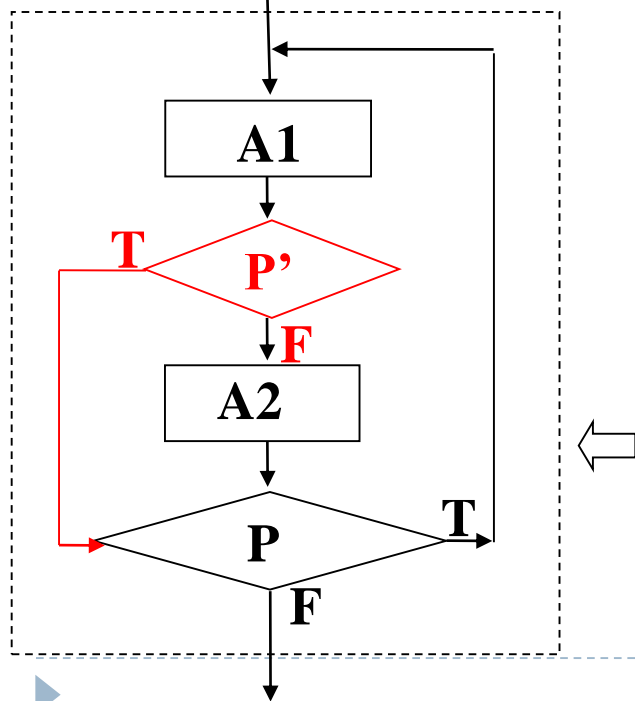
```
int d, sum = 0;
int i = 1;
while (i <= 10)
{
    scanf("%d", &d);
    if (d <= 0)
        continue;
    sum += d;
    i++;
}
printf("sum: %d \n", sum);
```



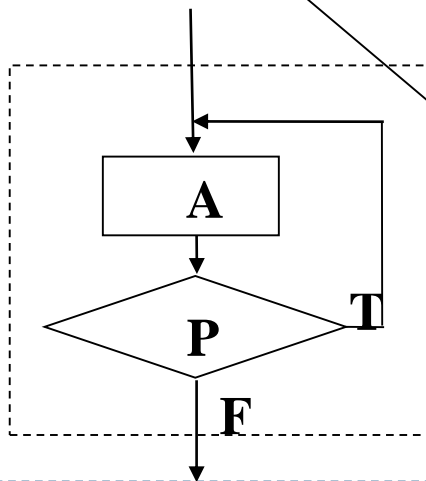
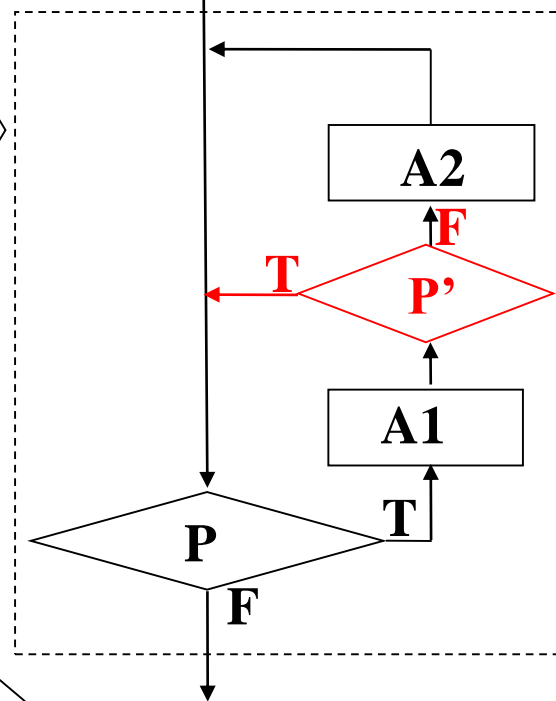
# continue “流程图”

等价于：

C语言中continue辅助do-while语句控制的循环流程



C语言中continue辅助while语句控制的循环流程



► 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("sum: %d \n", sum);
```

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



## 例：输入10个正数，求和；如果输入负数，则忽略该负数

等价于：

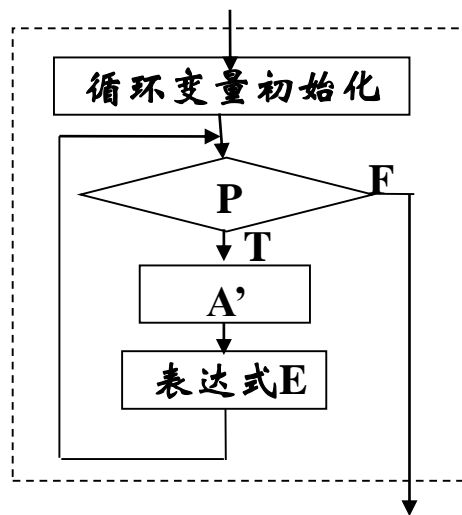
```
int d, sum = 0;
int i = 1;
while (i <= 10)
{
    scanf("%d", &d);
    if (d <= 0)
        continue;
    sum += d;
    i++;
}
printf("sum: %d \n", sum);
```

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

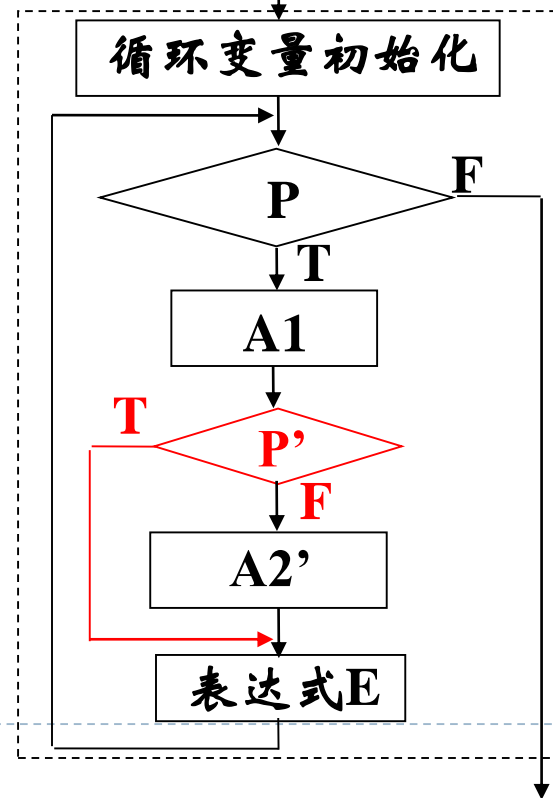
```
int d, sum = 0;
int i = 1;
do
{
    scanf("%d", &d);
    if (d <= 0)
        continue;
    sum += d;
    i++;
} while (i <= 10);
printf("sum: %d \n", sum);
```

- ▶ continue辅助for控制的流程 与  
continue辅助while/do-while控制的流程

“可能”不等价!



C语言中continue辅助  
for语句控制的循环流程



- 
- ▶ 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;  
        ...;  
    }  
    ...  
}
```



# 无条件转移控制

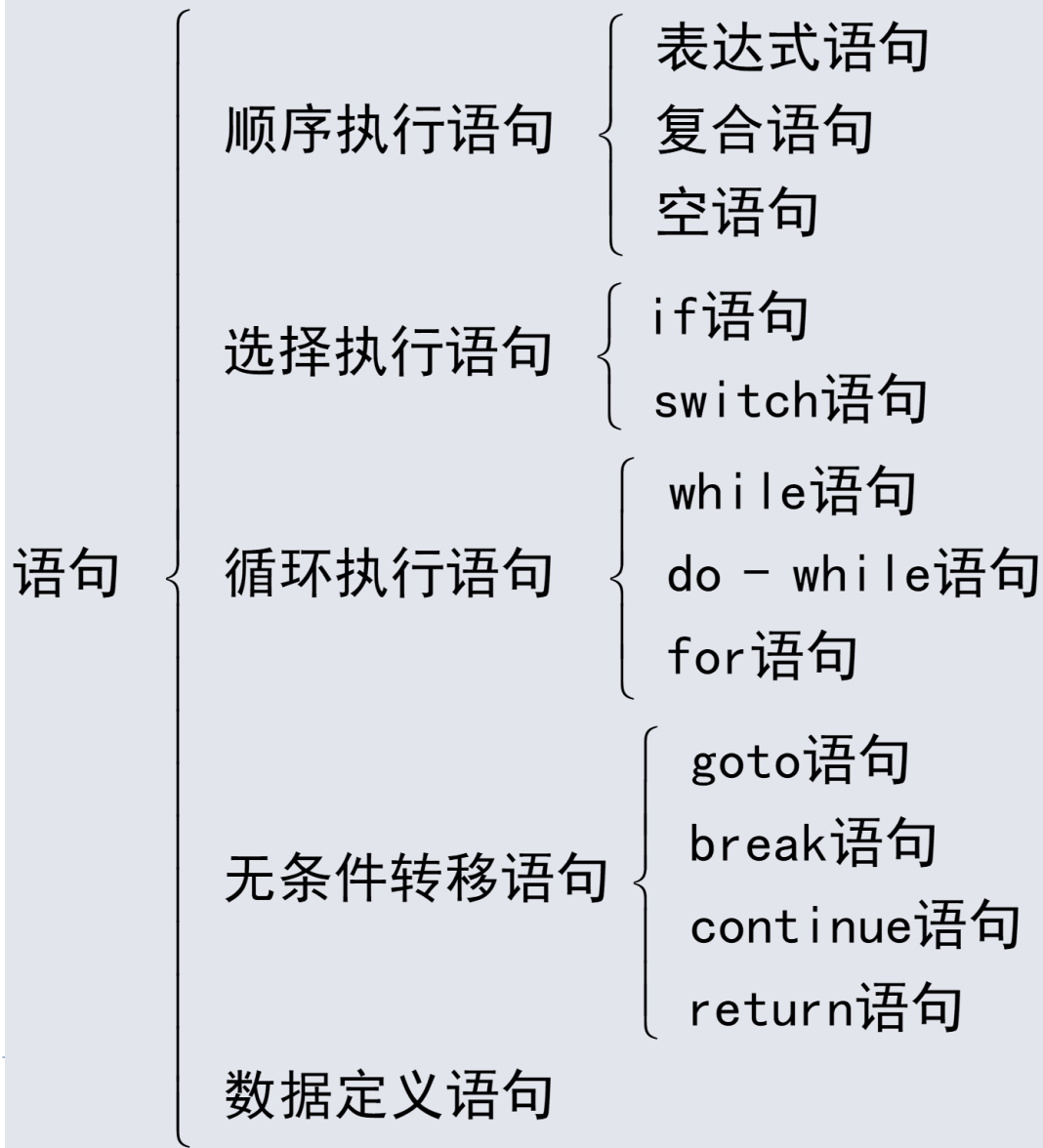
---

- ▶ if和switch是有条件的选择语句，相对于此：
  - ▶ goto (主要用于跳出多重循环)
  - ▶ break (switch; 循环折断)
  - ▶ continue (循环接续)
  - ▶ return (函数返回值)

以上语句是无条件转移控制语句！



# C语言的语句



## Q & A

---



# 内容回顾

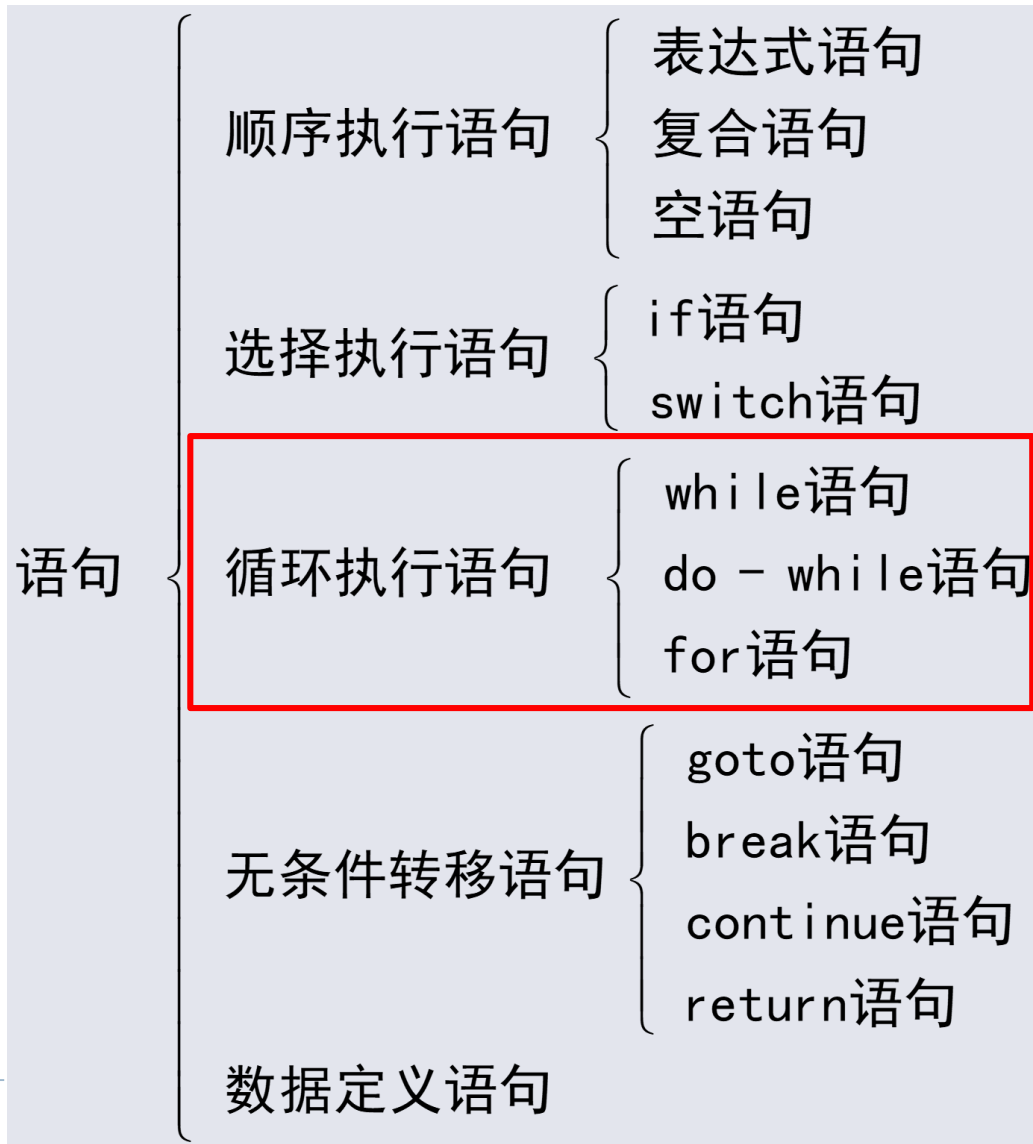
## 关于程序编译和调试

---

- ▶ 编译：把编译错误排除在Coding阶段
  - ▶ 每写完一个函数、甚至复合语句就F7编译一次
- ▶ 调试：
  - ▶ 断点调试
    - ▶ 普通断点、高级断点...
    - ▶ 在“可疑”的、关键地方F9设置断点，查看运行阶段变量值是否和预想（自己计算）的一致
    - ▶ F5, Ctrl+F5, F9, F10, F11...
  - ▶ cout输出
  - ▶ ...



## C/C++语句的分类





## 内容回顾

---

### ▶ 循环语句

#### ▶ while(循环条件)

```
{  
    <任务A>  
}
```

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

- ▶ 适合事件控制的循环
- ▶ 如果循环体至少执行一次，可使用do-while语句

#### ▶ for (初始条件;循环条件;表达式)

```
{  
    <任务A>  
}
```

- ▶ 适合计数控制的循环



例：计算从键盘输入的一系列整数的和，要求输入以0结束。  
(事件控制的循环)

## 内容回顾

```
#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。内容回顾

`while (a != 0)`

`while (i <= n)`



## 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，遇到输入的负数或0就提前终止求和的过程。

```
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
```

循环只执行了2次.

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

循环执行了4次.

# goto语句 - 无条件转移控制语句

内容回顾

- ▶ goto语句的格式如下：

goto <语句标号>; // (“代号”)

// <语句标号>为标识符，其定义格式为：

<语句标号>: <语句>



- ▶ goto的含义是：程序转移到特定<语句标号>的语句

## goto语句一个用途：跳出多重循环

```
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: ;
```



## goto语句一个用途：跳出多重循环

```
int i, j;  
for (i=1; i<=10; i++)  
    for (j=1; j<=10; j++)  
        if (i*j == 50)  
            goto END;  
END: cout << i << ", " << j << endl;  
//输出5, 10
```

```
int i, j;  
for (i=1; i<=10; i++)  
    for (j=1; j<=10; j++)  
        if (i*j == 50)  
            break;  
cout << i << ", " << j << endl;  
//输出11, 5
```