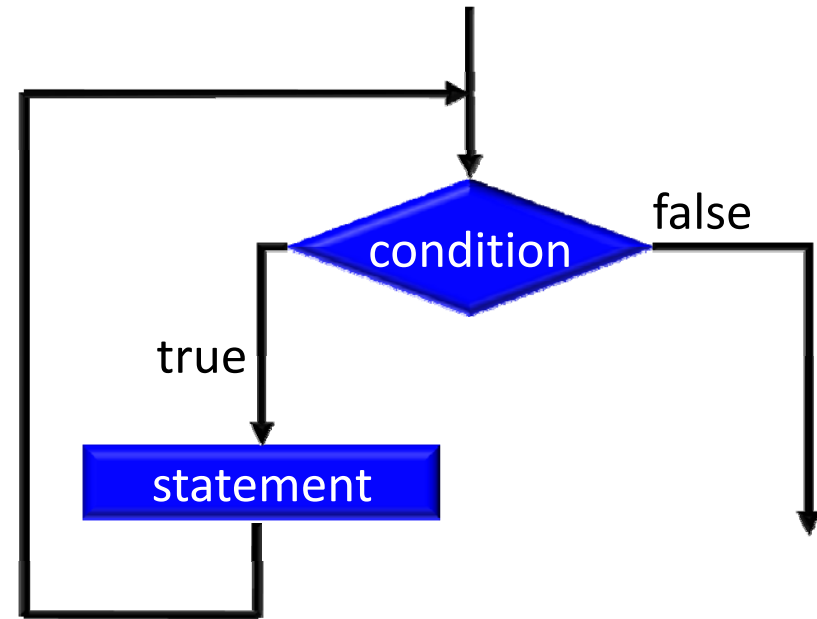
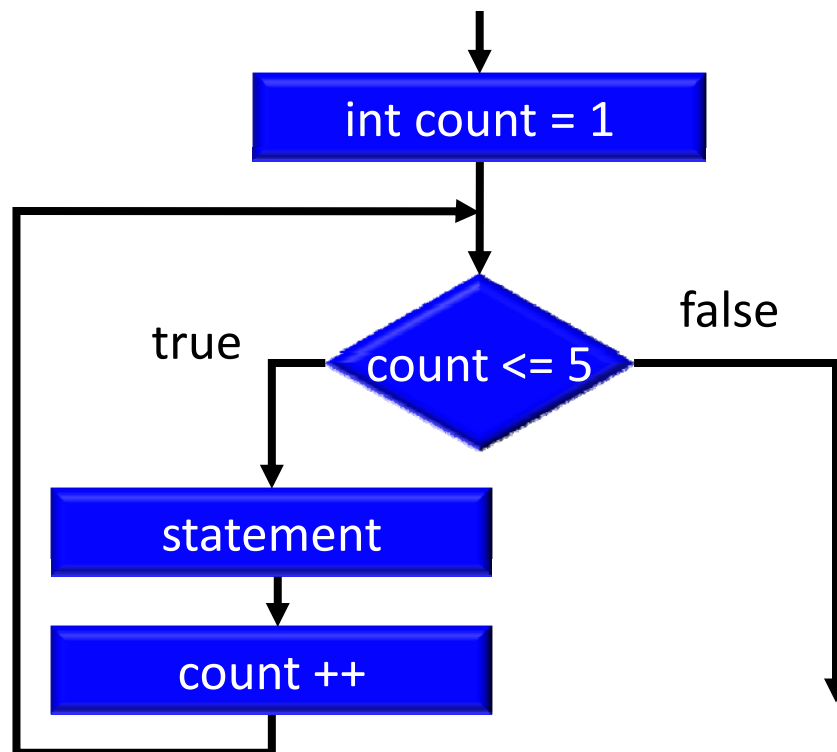


Loops and Repetition



First, the tax program

```
double income;
int filingStatus;
int numDependents;
int children;
double standardDeduction;
double deductions;
double taxableIncome;
double tax;

printf("Enter your annual income: ");
scanf("%lf", &income);

if(income < 9350)
{
    printf("You may be poor, but you owe no taxes\n");
    exit(0);
}
```

```

printf("What is your filing status? \n1) single\n");
printf("2) married filing jointly\n3) married filing separately\n");
printf("Please enter a number: ");
scanf("%d", &filingStatus);

switch(filingStatus)
{
    case 1:
        numDependents = 1;
        standardDeduction = 5700;
        break;

    case 2:
        printf("How many children do you have? ");
        scanf("%d", &children);
        numDependents = children + 2;
        standardDeduction = 11400;
        break;

    case 3:
        numDependents = 1;
        standardDeduction = 5700;
        break;

    default:
        printf("Invalid input!\n");
        exit(1);
        break;
}

```

```
deductions = standardDeduction + numDependents * 3650;
taxableIncome = income - deductions;

if(taxableIncome < 0)
{
    tax = 0;
}
else if(taxableIncome <= 16750)
{
    tax = taxableIncome * 0.10;
}
else if(taxableIncome <= 68000)
{
    tax = 1675 + 0.15 * (taxableIncome - 16750);
}
else if(taxableIncome <= 137300)
{
    tax = 9362.50 + 0.25 * (taxableIncome - 68000);
}
else
{
    tax = 26687.50 + 0.28 * (taxableIncome - 137300);
}

printf("%.2f\n", tax);
```

Loops and Repetition

Loops in programs allow us to repeat blocks of code.

Useful for:

- Trying again for correct input

- Counting

- Repetitive activities

- Programs that never end



Three Types of Loops/Repetition in C

- **while**
 - top-tested loop (pretest)
- **for**
 - counting loop
 - forever-sentinel
- **do**
 - bottom-tested loop (posttest)

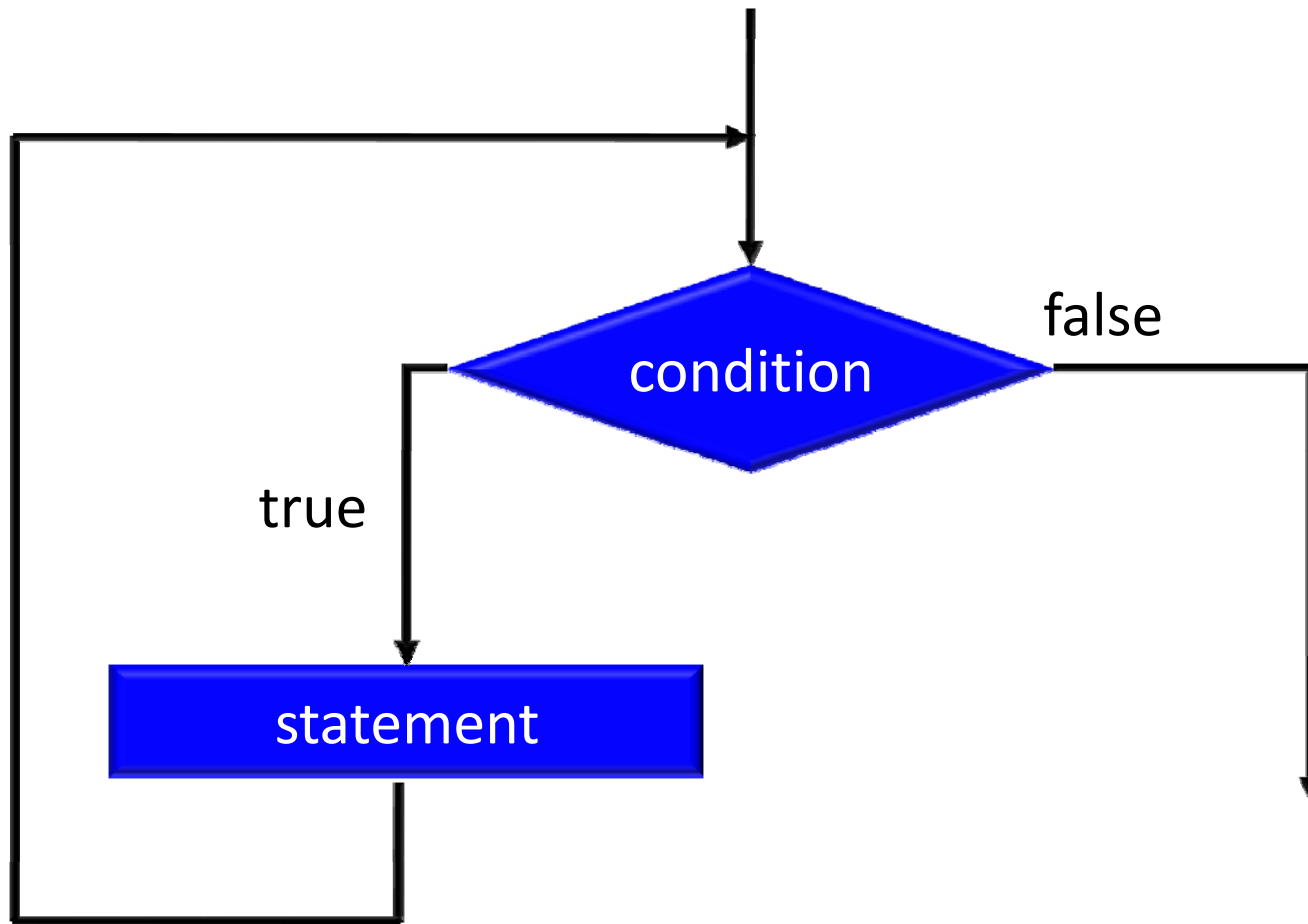
The **while** loop

Top-tested loop (pretest)

```
while (condition)  
    statement;
```

Note that, as in IF selection, only one statement is executed. You need a block to repeat more than one statement (using { })

while(condition)statement;



Similar to the if statement

- Check the boolean condition
- If true, execute the statement/block

Repeat the above until the boolean is false

Example

```
bool valid = true;           // Until we know otherwise

printf("Enter the inductance in millihenrys: ");
scanf("%lf", &l);

/* Test to see if the user entered an invalid value */
if(l <= 0)
{
    printf("You moron, you entered an invalid inductance!\n");
    valid = false;
}
else
    printf("Okay, I guess that's reasonable\n");
```

Remember this? What if we
input invalid values?

```
bool valid = false; /* Until we know otherwise */
while(!valid)      /* Loop until value is valid */
{
    printf("Enter the inductance in millihenrys: ");
    scanf("%lf", &l);

    /* Test to see if the user entered an invalid value */
    if(l < 0)
    {
        printf("You moron, you entered a negative inductance!\n");
    }
    else if(l == 0)
    {
        printf("You are really dumb, you entered zero.\n");
    }
    else
    {
        printf("Okay, I guess that's reasonable\n");
        valid = true;
    }
}
```

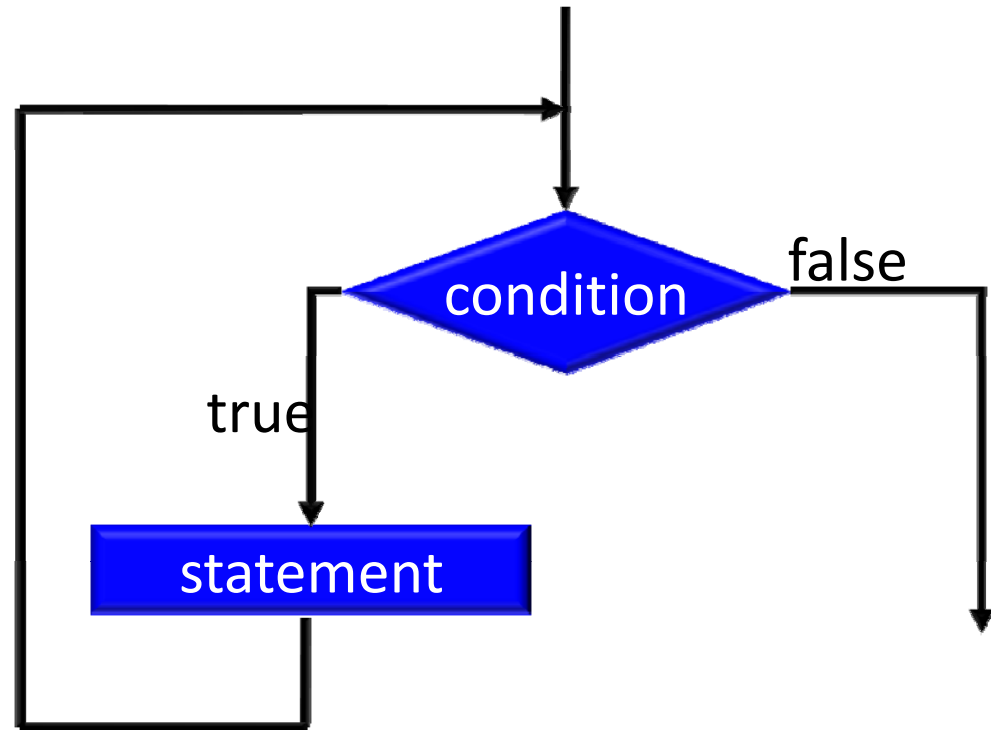
Example with while loop

What does this do different?

while

```
while (condition)
    statement;
```

```
while (condition)
{
    statement1;
    statement2;
}
```



```
while(!valid)      /* Loop until value is valid */
{
    printf("Enter the inductance in millihenrys: ");
    scanf("%lf", &l);

    if(l > 0)
    {
        valid = true;
    }
}
```

```
int i = 10;
while(i > 0)
{
    printf("i=%d\n", i);
    i = i - 1;
}
```



Forever loops and never loops

Because the conditional can be “always true” or “always false”, you can get a loop that runs forever or never runs at all.

```
int count=0;
while(count !=0)
    printf("Hi Mom");
```

What is wrong with
these statements?

```
while (count=1)    //insidious error!!!
    count = 0;
```

How to count using while

First, outside the loop, initialize the counter variable

Test for the counter's value in the boolean

Do the body of the loop

Last thing in the body should change the value of the counter!

```
i = 1;
while(i <= 10)
{
    printf("i=%d\n", i);
    i = i + 1;
}
```

The **for** loop

The **while** loop is pretty general. Anything that can be done using repetition can be done with a while loop

Because counting is so common, there is a specialized construct called a for loop.

A for loop makes it easy to set up a counting loop

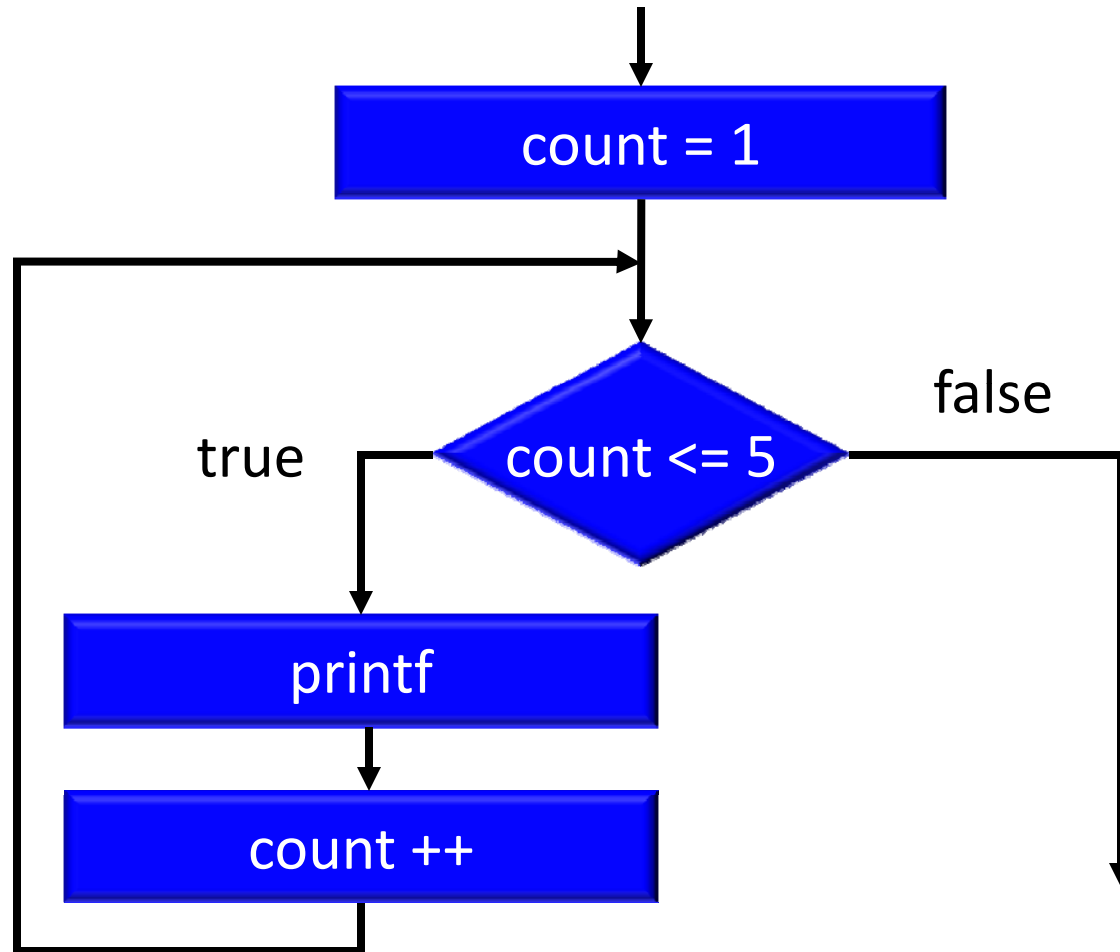
Three parts

```
for(count=1; count<=5; count++)  
    statement;
```

Three parts to a for loop (just like the while):

- Set the initial value for the counter
- Set the condition for the counter
- Set how the counter changes each time through the loop

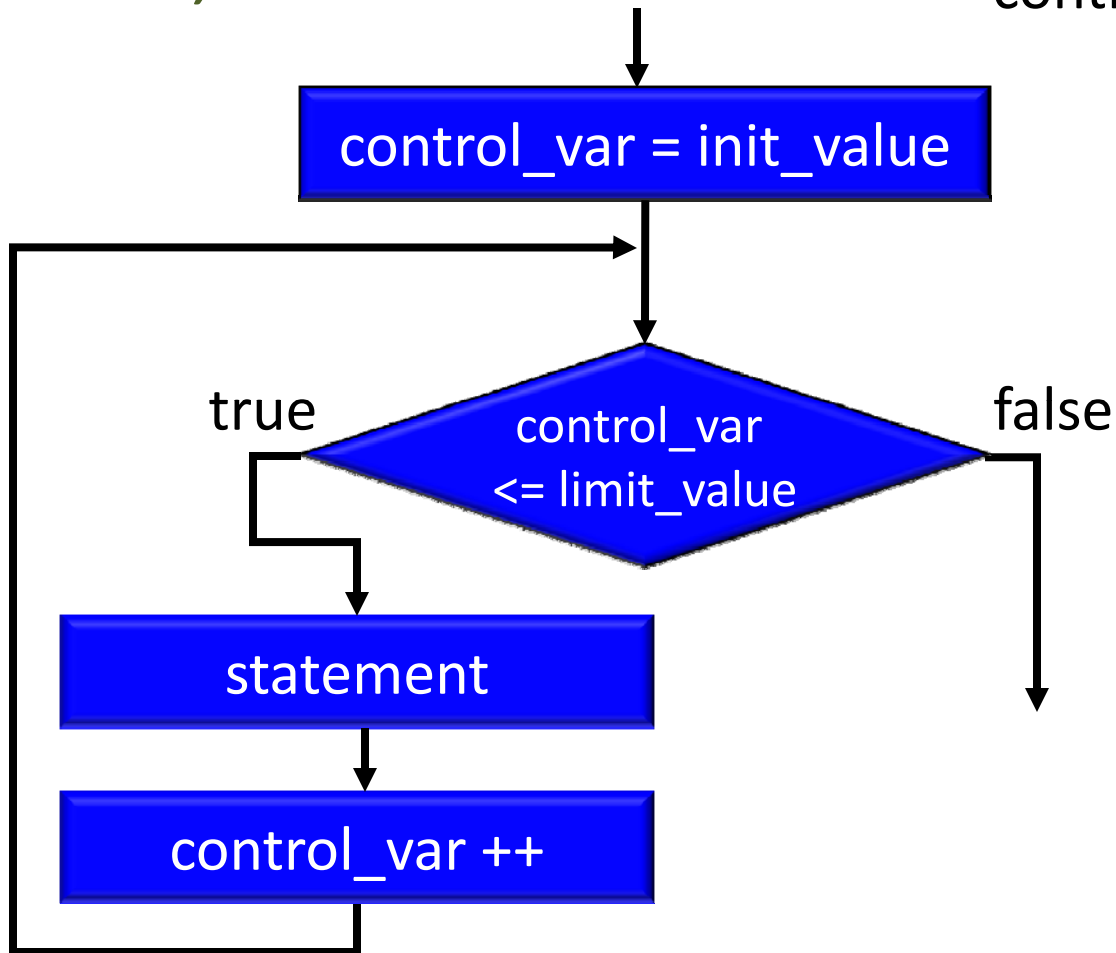

```
for(count=1; count<=5; count++)  
    printf("count=%d\n", count);
```



Ascending **for**

<=, ++

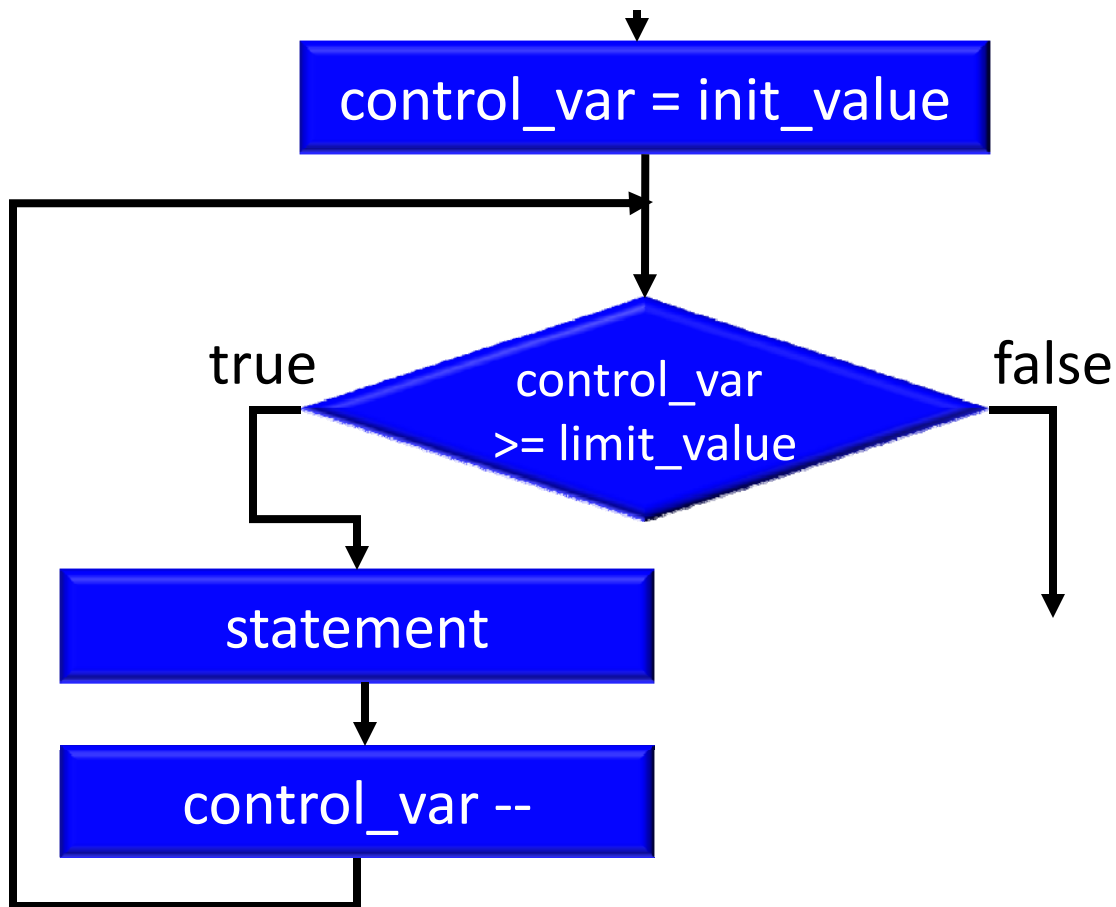
```
for (control_var=init_value;  
    control_var <=limit_value;  
    control_var++) statement;
```



Descending for

$\geq, --$

```
for (control_var=init_value;  
    control_var >=limit_value;  
    control_var--) statement;
```



Comments

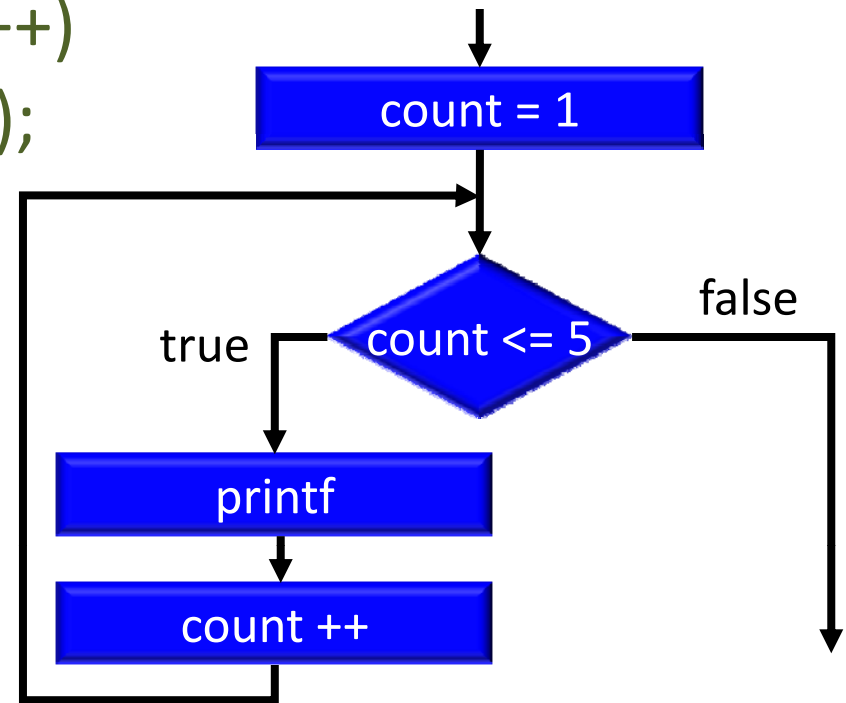
- It is dangerous to alter `control_var` or `limit_var` within the body of the loop.
- The components of the `for` statement can be arbitrary statements, e.g. the loop condition may be a function call.

```
for(count=1; count<=5; count++)  
    printf("count=%d\n", count);
```

```
for(i=1; i<=10; i++)  
{  
    printf("%d\n", i);  
}
```

```
for(t = 1.7; t < 3.5; t = t + 0.1)  
{  
    printf("%f\n", t);  
}
```

```
for(i=1; i<5; i++)  
{  
    for(j=1; j<4; j++)  
    {  
        printf("%d * %d = %d\n", i, j, i * j);  
    }  
}
```



Top-tested Equivalence

The following loop

```
for(x=init; x<=limit; x++)  
    statement_list
```

is equivalent to

```
x=init;  
while (x<=limit){  
    statement_list;  
    x++;  
}
```

Some Magic Statements

`s += 12;` `/* Equivalent to s = s + 12; */`

`s -= 13;` `/* Equivalent to s = s - 13; */`


These work fine for
integers or floating point

break;

The break statement exits the containing loop immediately!

```
while(true)          /* Loop until value is valid */
{
    printf("Enter the inductance in millihenrys: ");
    scanf("%lf", &l);

    /* Test to see if the user entered an invalid value */
    if(l <= 0)
    {
        printf("You moron, you entered an invalid inductance!\n");
    }
    else
    {
        printf("Okay, I guess that's reasonable\n");
        break;
    }
}
```



The do/while loop

Often just called a “do loop”.

- **do/while**

- bottom-tested loop (posttest)

```
do
{
    angle += 2 * M_PI / 20;
    sinVal = sin(angle);
    printf(“sin(%f) = %f\n”, angle, sinVal);
} while(sinVal < 0.5);
```

Bottom-tested Loop: **do**

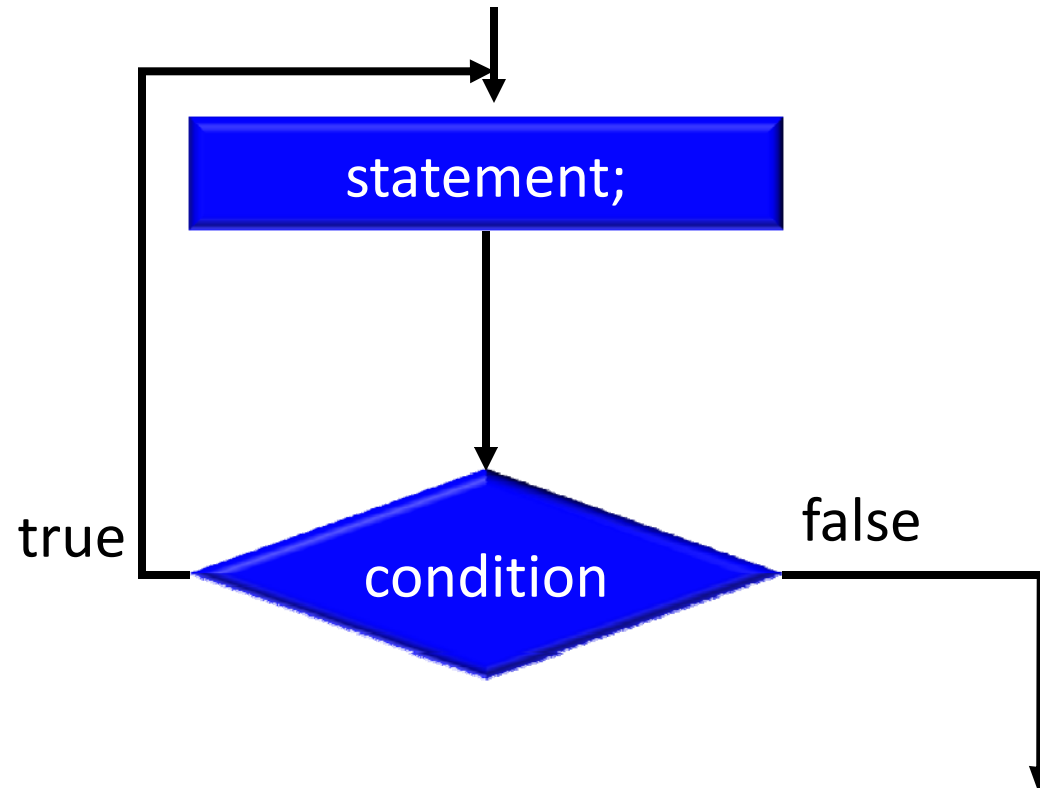
- Bottom-tested (posttest)
- One trip through loop is guaranteed, i.e. **statement** is executed at least once

```
do  
    statement  
while (loop_condition);
```

```
do  
{  
    statement1;  
    statement2;  
}  
while (loop_condition);
```

Usually!

```
do { statement; } while(condition)
```



do/while Examples

```
angle = M_PI / 2;
do
{
    angle -= 0.01;
    cosVal = cos(angle);
    printf("cos(%f)=%f\n", angle, cosVal);
} while(cosVal < 0.5);
```

```
i = 0;
do
{
    i++;
    printf("%d\n", i);
} while(i < 10);
```

```
do
{
    printf("Enter a value > 0: ");
    scanf("%lf", &val);
} while(val <= 0);
```

Bottom-tested Equivalence

- Bottom-tested do loop (posttest)

```
do
{
    statement;
}
while (condition);
```

- Similar to bottom-tested forever loop

```
for (;;)
{
    statement_list;
    if (!condition) break;
}
```

The “one off” error

It is easy to get a for loop to be “one off” of the number you want. Be careful of the combination of init_value and < vs. <=

Counting from 0, with <, is a good combination and good for invariants as well.

```
for(i=1; i<10; i++)  
{  
}
```

```
for(i=1; i<=10; i++)  
{  
}
```

```
for(i=0; i<10; i++)  
{  
}
```

The “one off” error

It is easy to get a for loop to be “one off” of the number you want. Be careful of the combination of init_value and < vs. <=

Counting from 0, with <, is a good combination and good for invariants as well.

```
for(i=1; i<10; i++)  
{  
    9 values: 1 to 9  
}
```

```
for(i=1; i<=10; i++)  
{  
    10 values: 1 to 10  
}
```

```
for(i=0; i<10; i++)  
{  
    10 values: 0 to 9  
}
```

while, for, do/while

```
for(t = 1.7; t < 3.5; t = t + 0.1)
{
    printf("%f\n", t);
}
```

```
i = 0;
do
{
    i++;
    printf("%d\n", i);
} while(i < 10);
```

```
for(i=1; i<5; i++)
{
    for(j=1; j<4; j++)
    {
        printf("%d * %d = %d\n", i, j, i * j);
    }
}
```

```
while(!valid) /* Loop until val is valid */
{
    printf("Enter the inductance in millihenrys: ");
    scanf("%lf", &l);

    if(l > 0)
    {
        valid = true;
    }
}
```

```
angle = M_PI / 2;
do
{
    angle -= 0.01;
    cosVal = cos(angle);
    printf("cos(%f)=%f\n", angle, cosVal);
} while(cosVal < 0.5);
```

```
for(i=1; i<=10; i++)
{
    printf("%d\n", i);
}
```

```
do
{
    printf("Enter a value > 0: ");
    scanf("%lf", &val);
} while(val <= 0);
```

```
int i = 10;
while(i > 0)
{
    printf("i=%d\n", i);
    i = i - 1;
}
```

