CISC 105
Arithmetic, User Input, Conditionals, Looping

June 13, 2005

Announcements

- Expect you to check your email regularly
  - Check the web site too
  - Reading assignments
- Lab 00
- Questions?

Review

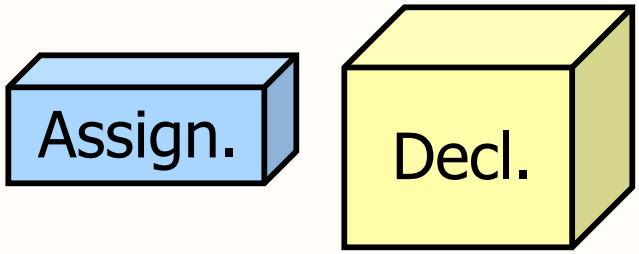
- Programming process
- Program structure
  - stdio.h, main, comments
- Data types: int, float, double, char
- Variable declarations, initializations
- Printing variables-format specifiers
- Arithmetic
  - Integer division vs. double division

Time for the first quiz!

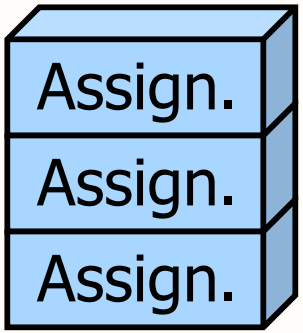
Review quiz

Building Blocks

- Each of these statements is a building block
  - Declaration
  - Assignment



- We can combine them to create more complex programs



- Introducing more building blocks today

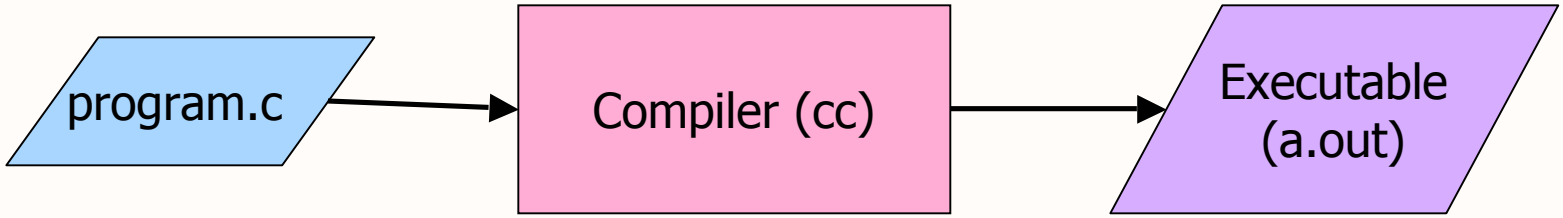
Constants

- Replace all instances of an identifier with a value
  - Example: #define PI 3.14159
  - Use **PI** in program instead of typing all of number
  - Does not change throughout execution of program
- Benefits
  - Change one value (in #define) to change value everywhere in program
  - Flexible programs
  - Gets rid of magic numbers

Constants

- #define SYMBOLIC_NAME replacement
  - **Preprocessor directive**
  - Before main in program

Note: no ';'



Compiler: replaces each
SYMBOLIC_NAME with replacement

- Style: typically, identifier is ALL CAPS
  - Gets replaced throughout program, so need to be careful with naming

Constants

- Don't use constants on left hand side of statements
  - Can't change value

```
#define PI 3.14159
...
   PI = 3.141;
```

ERROR: compiler attempts to replace PI with 3.14159, i.e.,

3.14159 = 3.141;

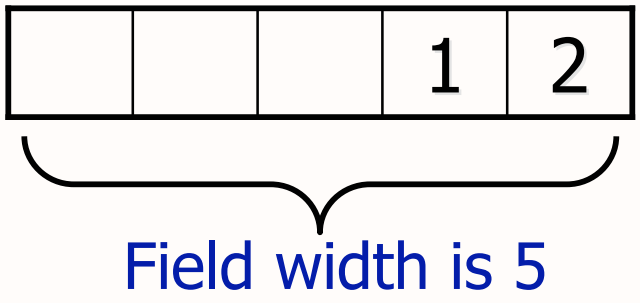
Format Specifiers

- More formatting options for printf
- For integers: %[fieldwidth]d
- For floats, doubles
  - %[fieldwidth][.precision]f (or lf)
- Fieldwidth
  - Minimum number of character spaces reserved to display the entire value
  - Includes decimal point, digits before and after the decimal point and the sign
- Precision
  - Maximum number of digits after the decimal point

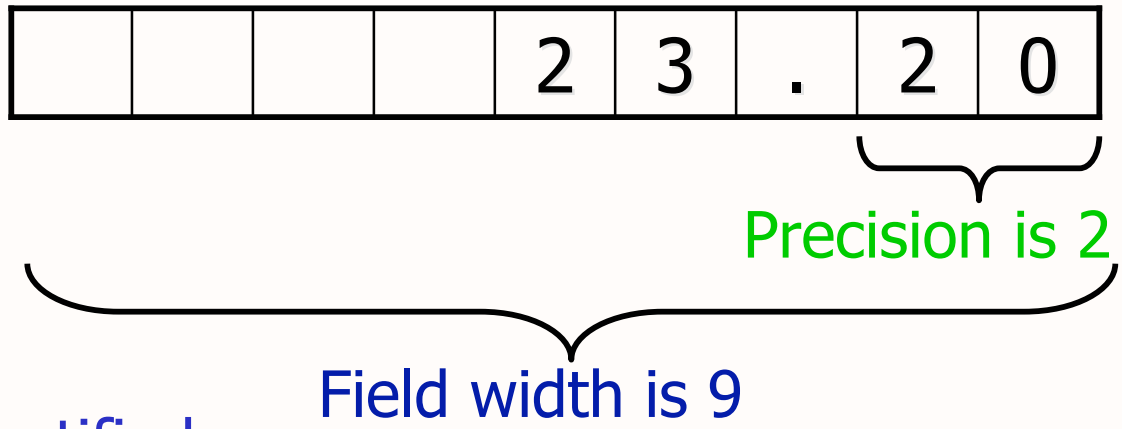
The [] mean "optional"

Format Specifiers

printf("%5d", month);



printf("%9.2f", expense);



- What if precision is bigger than the decimal places? Fills decimal with 0s
- What if field width is smaller than the length of the value? Prints entire part before the decimal point (for float, double); all for integer

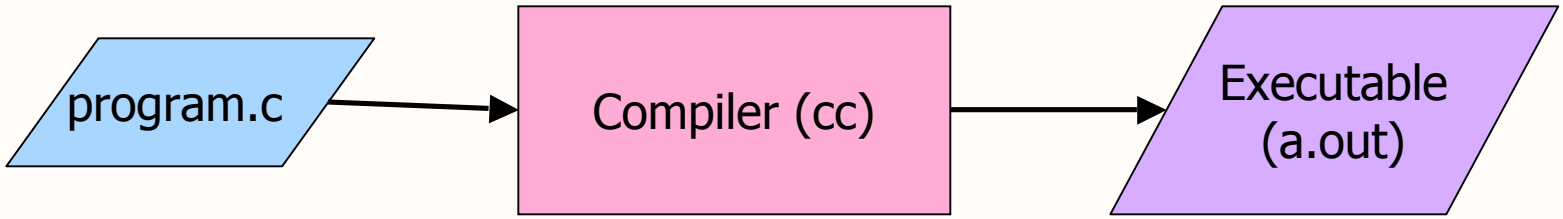
Math Library

- Commonly used math functions
  - pow: raise a number to a power
  - sin: find the sine of an angle, in radians
  - cos: find the cosine of an angle, in radians
  - floor: round down to the nearest whole number
  - ceil: round up to the nearest whole number
  - log: find the log of a number
  - exp: raise e to a power
  - sqrt: find the square root of a number
- To learn more about a particular function

  > man function_name

Using the Math Library

- Include math.h
- Compile with the flag "lm"



cc -lm program.c

- Practice:
  - The number of cartons for X eggs
  - Find the sine, cosine of an angle of Y degrees

User Input

- We wrote a nice, general program, but we assigned specific values
- Want to allow the user to set the values
- Add flexibility to our programs

- > ./a.out

  Welcome!

  This program converts temperatures in Fahrenheit into Celsius.

  What temperature would you like to convert? 98.6

  98.6 degrees F is 37.0 degrees C.

User input

User Input: scanf

- Using **scanf** is similar to using printf
  - scanf("string literal", arguments);
- For each format specifier in string literal, must pass an argument
- Example: scanf("%d:%d", &hour, &minute);
- Differences from printf
  - How parameters are passed
  - Don't use escape characters in string literal
  - Minimize non-format-specifiers in string literal

User Input: scanf

- scanf is similar to assignment statements
  - After executing scanf, variables are assigned the values of the user's input

- Examples
  - Add user input to temperature converter
  - Add user input to average program

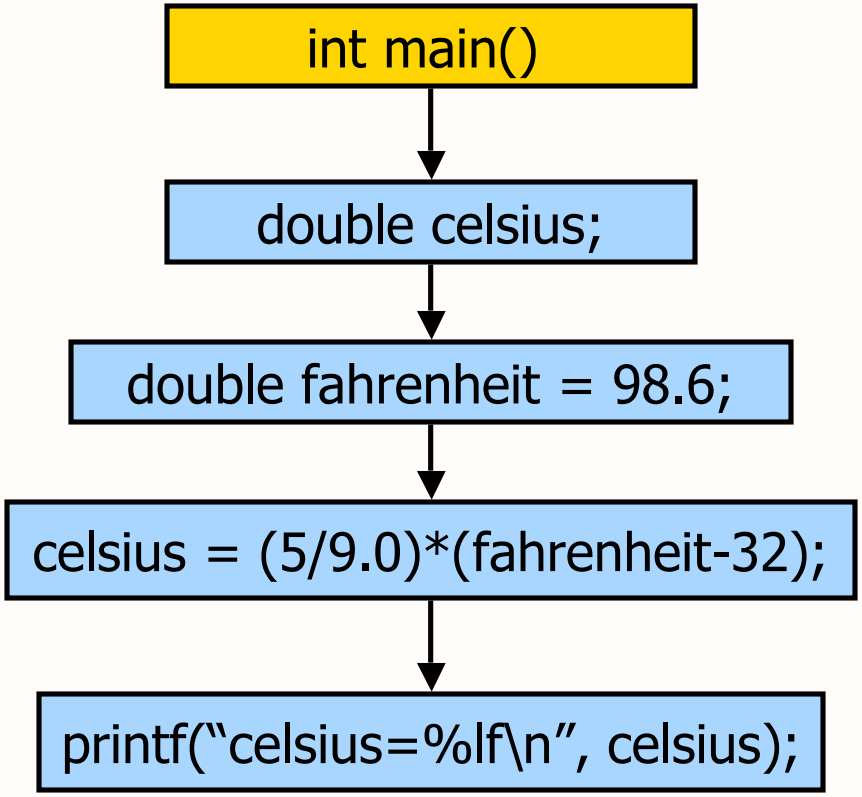
Conditionals

- Sometimes, we only want to execute a statement in certain cases
  - Example: Finding the absolute value of a number
    - |4| = 4
    - |-10| = 10
  - To get the answer, we only want to multiply the number by -1 if it's a negative number
  - Code:

```
if( x < 0 ) {
        x *= -1;
}
```

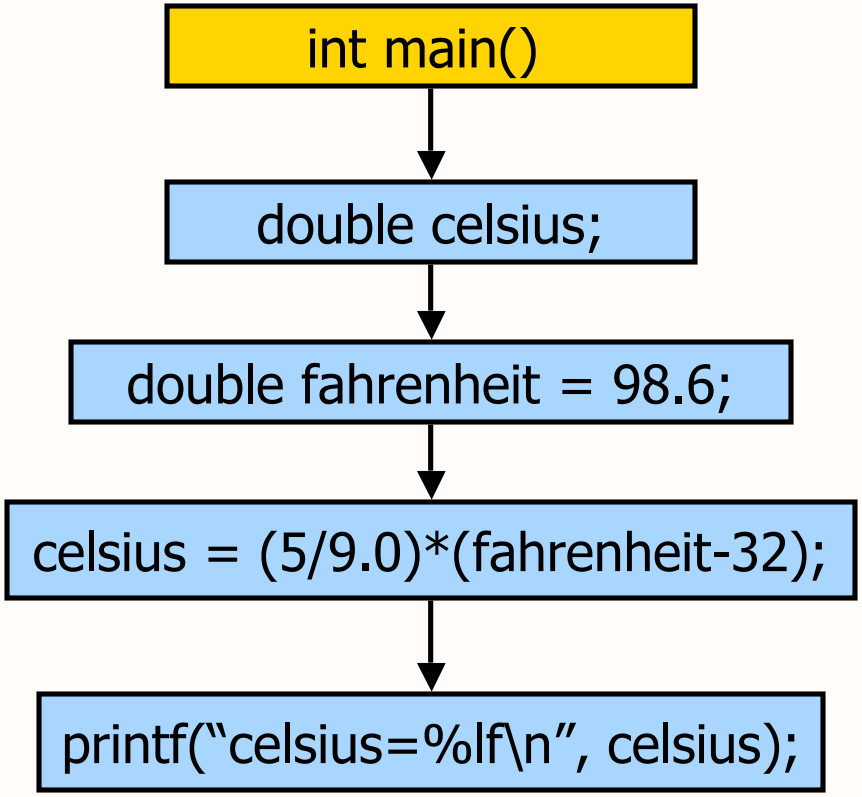
Conditional Statements

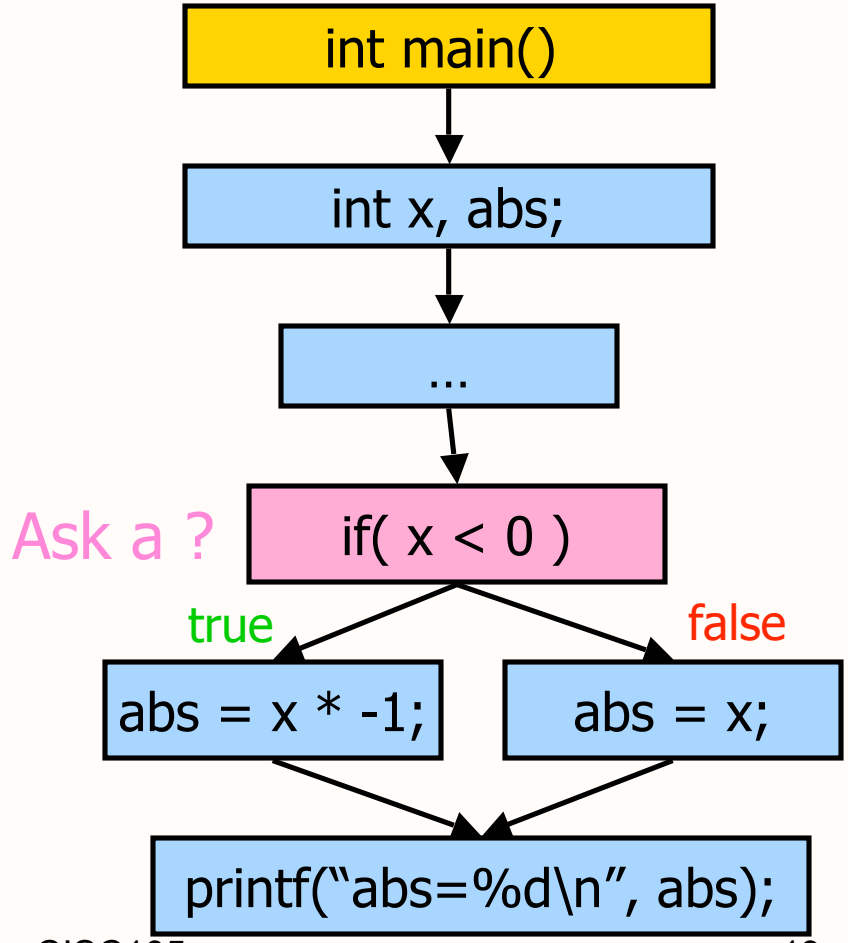
- Change the **control flow** of the program



Conditional Statements

- Change the **control flow** of the program





Syntax of **if** statement

```
if( condition ) {
    statements;
}
```

Don't need the {} if only one statement to execute when condition is true

Comparison Operators

- Use comparison operators to change the program's flow

Operator	Meaning
<	Less than?
<=	Less than or equal to?
>	Greater than?
>=	Greater than or equal to?
==	Equals?
!=	Not equals?

Examples: Using Conditionals

- Determine if a number is even or odd

```
int x;
printf("Enter a number: ");
scanf("%d", &x);
if( x%2 == 0 ) {
   printf("%d is even\n", x);
}
if( x%2 == 1 ) {
   printf("%d is odd\n", x );
}
```

Assignment operator vs. Equality operator

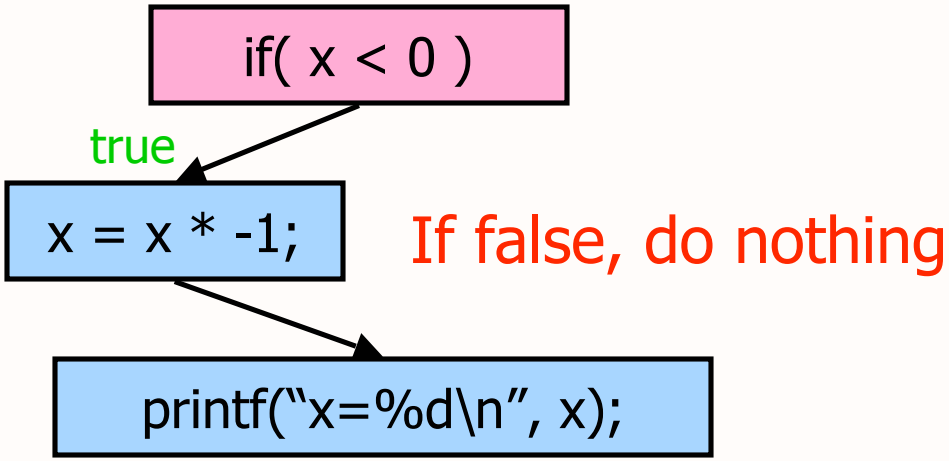
- Assignment operator: =
- Equality operator: ==

```
int x, remainder;
...
remainder = x%2;
if( remainder = 0 ) {
   printf("%d is even.\n", x);
}
```

Logic error
But will compile

If-Else statements (absolute values)

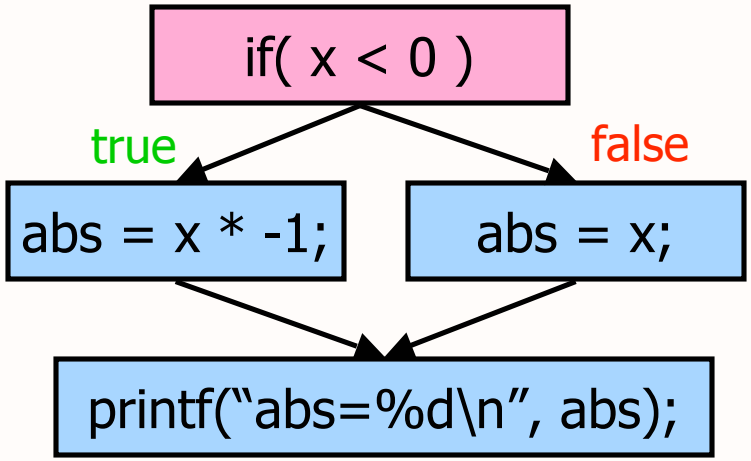
If statement



If false, do nothing

```
if( x < 0 ) {
        x *= -1;
}
```

If-else statement




```
if( x < 0 ) {
        abs = x * -1;
}
else {
        abs = x;
}
```

Syntax of **if-else** statement

```
if( condition ) {
    statements;
}
else {
    statements;
}
```

Don't need the {} if only one statement to execute when condition is true

Using the building blocks: nesting if-else statements

```
if( condition ) {
        if( condition ) {
                statements;
        }
        else {
                statements;
        }
}
else {

}
```

if-else statement is **nested** inside the if

Using the building blocks: nesting if-else statements

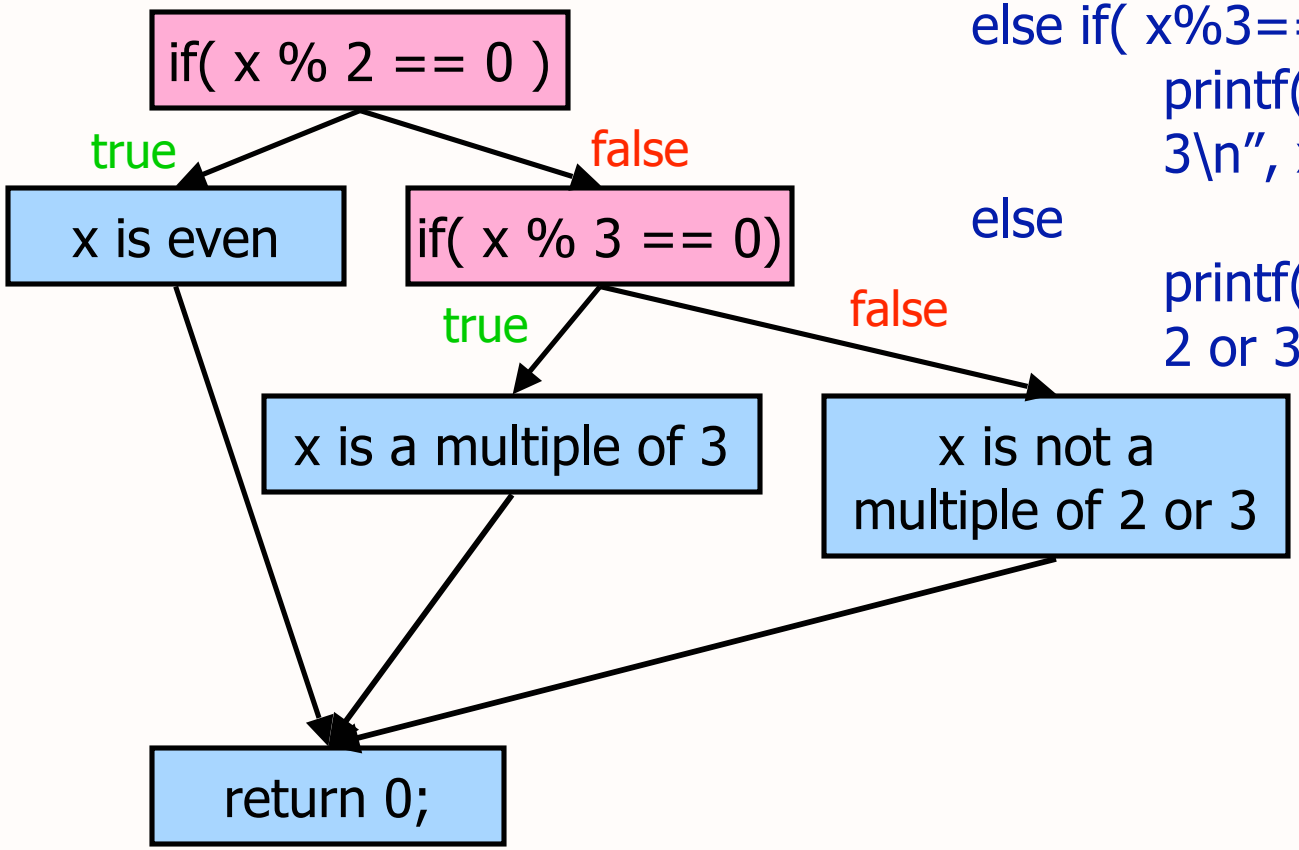
```
if( condition ) {
    statements;
}
else {
        if( condition ) {
                statements;
        }
        else {
                statements;
        }
}
```

if-else statement is **nested** inside the else

This structure can be rewritten as an if-else-if statement

If-Else-If statements

```
if( x %2 == 0 )
        printf("%d is a multiple of
        2\n", x);
else if( x%3==0 )
        printf("%d is a multiple of
        3\n", x);
else
        printf("%d is not a multiple of
        2 or 3\n", x);
```



Syntax of **if-else-if** statement

```
if( condition ) {
    statements;
}
else if( condition ) {
    statements;
}
else {
    statements;
}
```

Can have more than one of these else-if conditions

else is always optional; use when you want to do something that is not covered by the other conditions

More Complex Conditions

- Boolean
  - Dealing with the two logical values: true (1) and false (0)
- Combine conditions with Boolean operators
  - && (AND) – true only if both operands are true
  - || (OR) – true if at least one operand is true
  - ! (NOT) – true if the operand is not true

Truth Tables

operands

A	B	A&&B	A||B	!A	!B	!A &&B	A||!B
T	T	T	T	F			
T	F	F	T	F			
F	T	F	T	T			
F	F	F	F	T			

How does C evaluate conditionals?

- Assigns values to the conditionals
  - False: 0
  - True: non-zero
- Example:

```
int x = 2, y = 3, z;
z = x < y;
if( z == 0 )
   printf("x is greater than or equal to y.\n");
else
   printf("x is less than y.\n");
```

Example

```
int x = 2;
int y = 3;
int z = 4;
int b = x==2;
int c = !b;
int d = (y<4) && (z<3);
d = (y<4) || (z<3);
d = !d;
```

Practice: Verify if we're right by writing a program that outputs the correct values.

Switch Statement

- Meaning: multiple if-else-if statements with equals as the comparison operator
- Syntax:

```
switch( expr ) {
    case c1: actions; break;
    case c2: actions; break;
   ...
    default: actions;
}
```

Means: if expr == cn, do the associated actions

Switch Statement

- Meaning: multiple if-else-if statements with equals as the comparison operator
- Syntax:

expr's data type is either int or char

```
switch( expr ) {
    case c1: actions; break;
    case c2: actions; break;
    ...
    default: actions;
}
```

unique values

actions, breaks are optional

break means that terminate execution in the switch statement

Example using switch: Menu options

```
int option;
/* print menu to user, get user input */
switch( option ) {
   case 1: /* convert celsius to fahrenheit */
    break;
   case 2: /* convert fahrenheit to celsius */
    break;
   default: /* option not understood */
}
```

Example using switch: Menu options

```
char option;
/* print menu to user, get user input */
switch( option ) {
   case 'c': /* convert celsius to fahrenheit */
    break;
   case 'f': /* convert fahrenheit to celsius */
    break;
   default: /* option not understood */
}
```

Why break is optional

- Why would we not want to use break?
  - When multiple cases should do the same action
- Example:

```
int x, answer;
...
answer = x / 10;
switch( answer ) {
    case 0: printf("%d is less than 10.\n", x); break;
    case 1:
    case 2: printf("%d is less than 30.\n", x);break;
    default: printf("%d is greater than or equal to 30.\n", x);
     break;
}
```

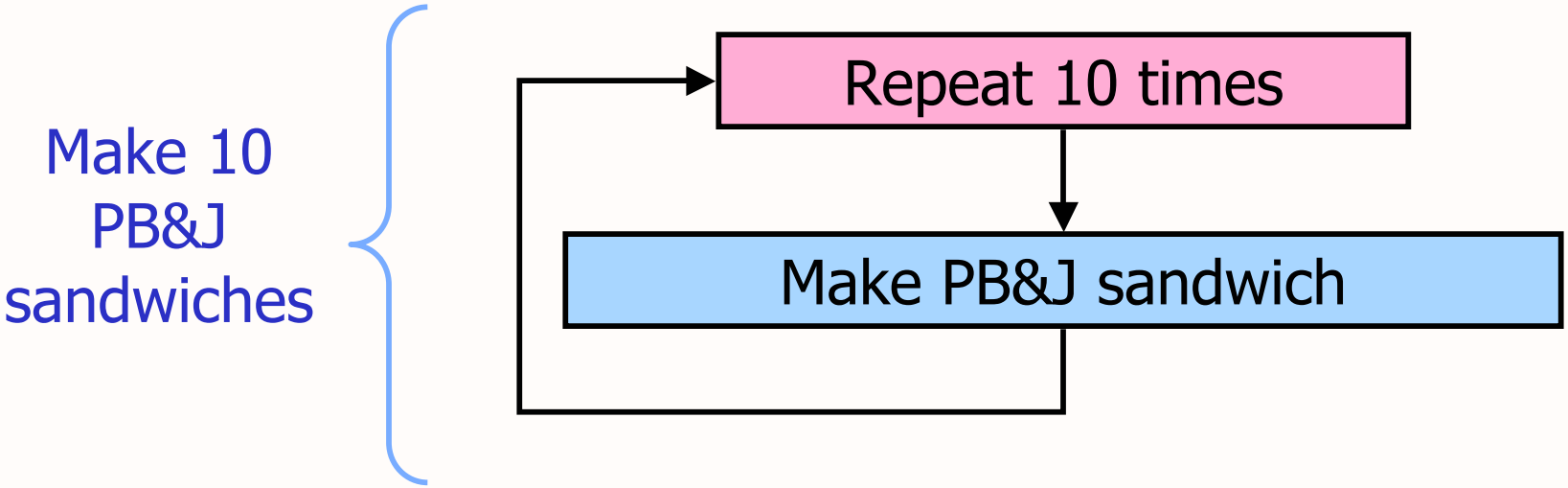
Switch or If-Else-if?

- When should you use switch vs. if-else-if statements?

- Practice: write menu with if-else-if statements and compare
  - Which is easier to read and understand?

Looping

- Iterative control structures



While Loop Syntax

```
while( condition ) {
    statements;
}
```

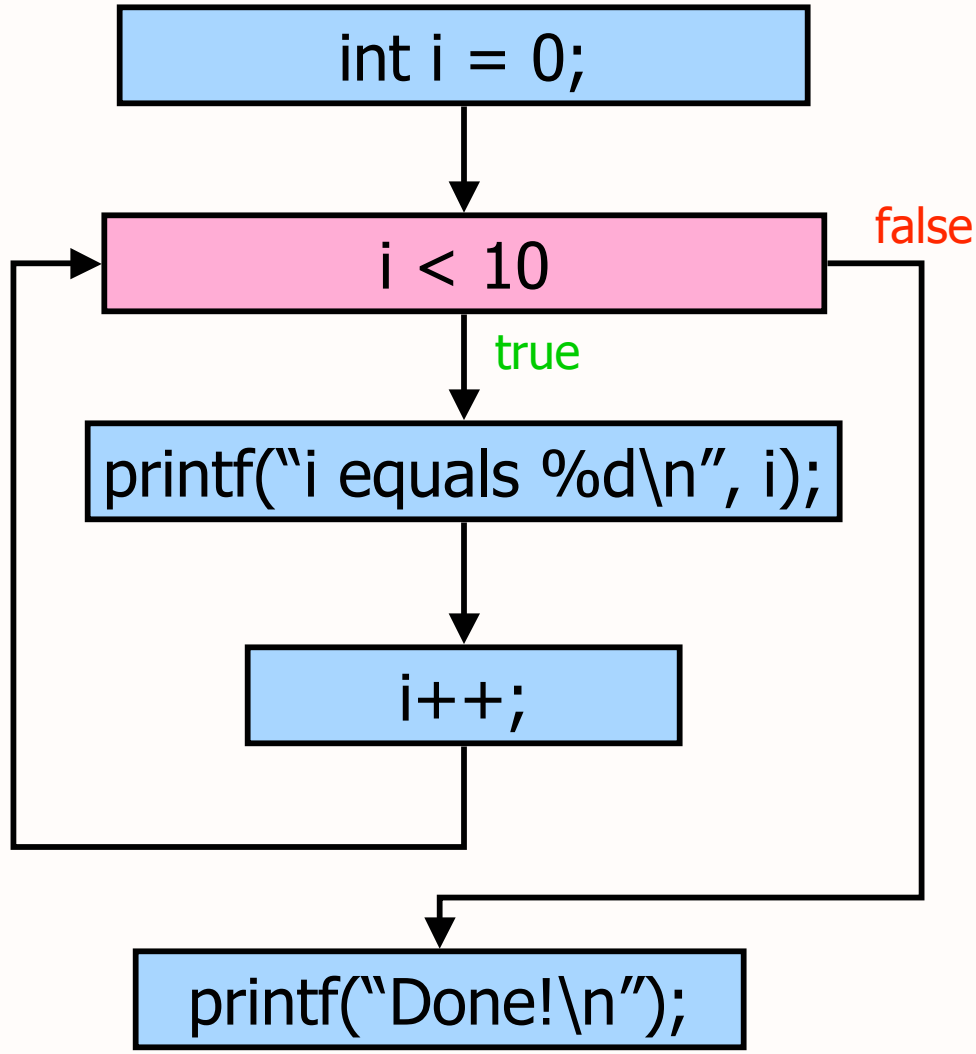
Braces are optional if only one statement

Also called body of while loop

- Like a repeated if statement.
  - Execute statements only when condition is true

While Loop

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



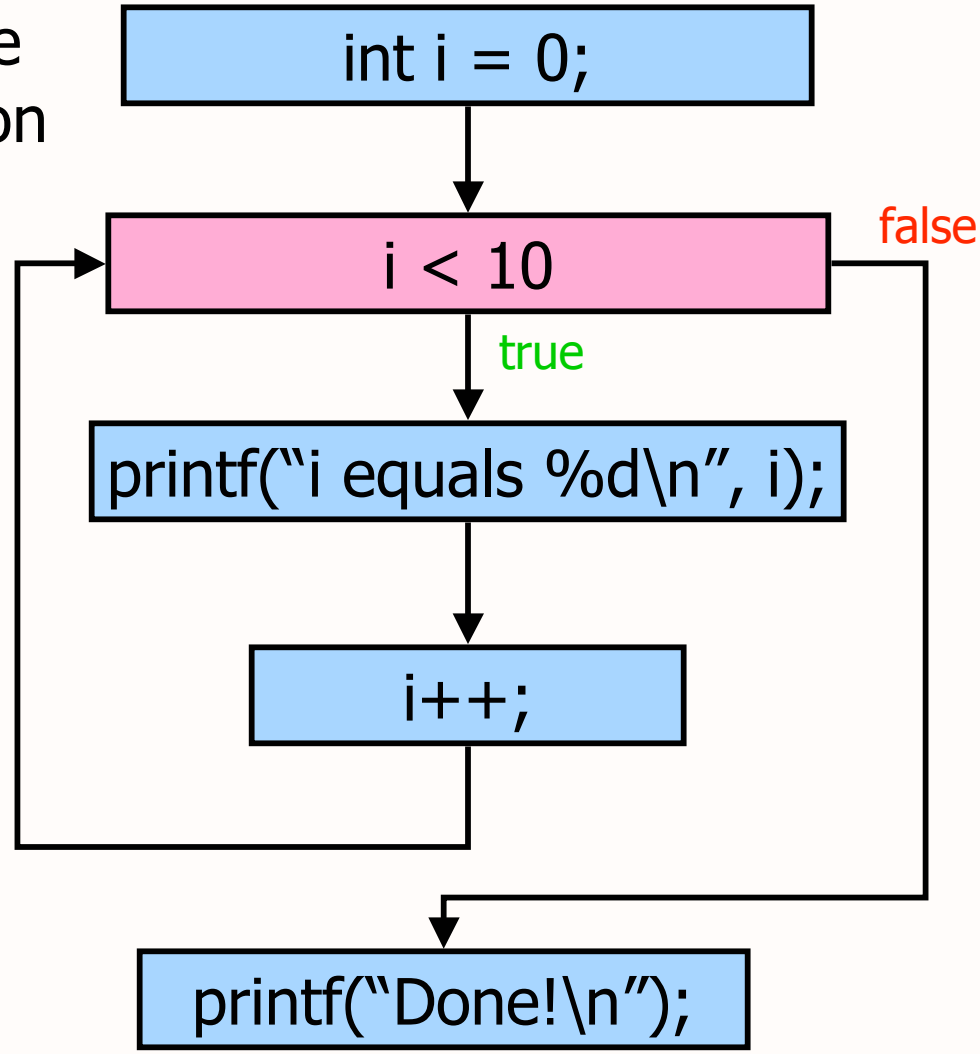
Questions:

- How many times will "i" get printed out?
- How many times is the condition evaluated?

While Loop

Initialize i before using in condition

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



Questions:

- How many times will "i" get printed out?
- How many times is the condition evaluated?

While Loop

- Question:
  - How can we make sure that the loop actually stops?
- Examples:
  - Add looping to temperature converter.
  - Prompt for input until user enters an even number
  - Printing a 10x10 square

Comment about Book

- In section 4.8, the authors show how to stop a loop using the break statement
  - This is UCKY code
    - Better programmers could write the code in a cleaner, more readable way
  - However, you should know how to read the code in case you see it in someone else's code.
  - I do not want you to write ucky code in this course.

While Loops, comparing use of break

```
/* condition shows when loop
   will stop executing */
while( x % 2 != 0 ) {
    printf("Enter a number");
    scanf("%d", &x);
}
printf("%d is an even
    number.\n", x);
```

```
/* have to look inside loop to
   know when it stops */
while( i ) {
    printf("Enter a number");
    scanf("%d", &x);
    if( x %2 == 0 ) {
        break;
    }
}
printf("%d is an even
    number.\n", x);
```

Do-while Loop

- Variation of while loop

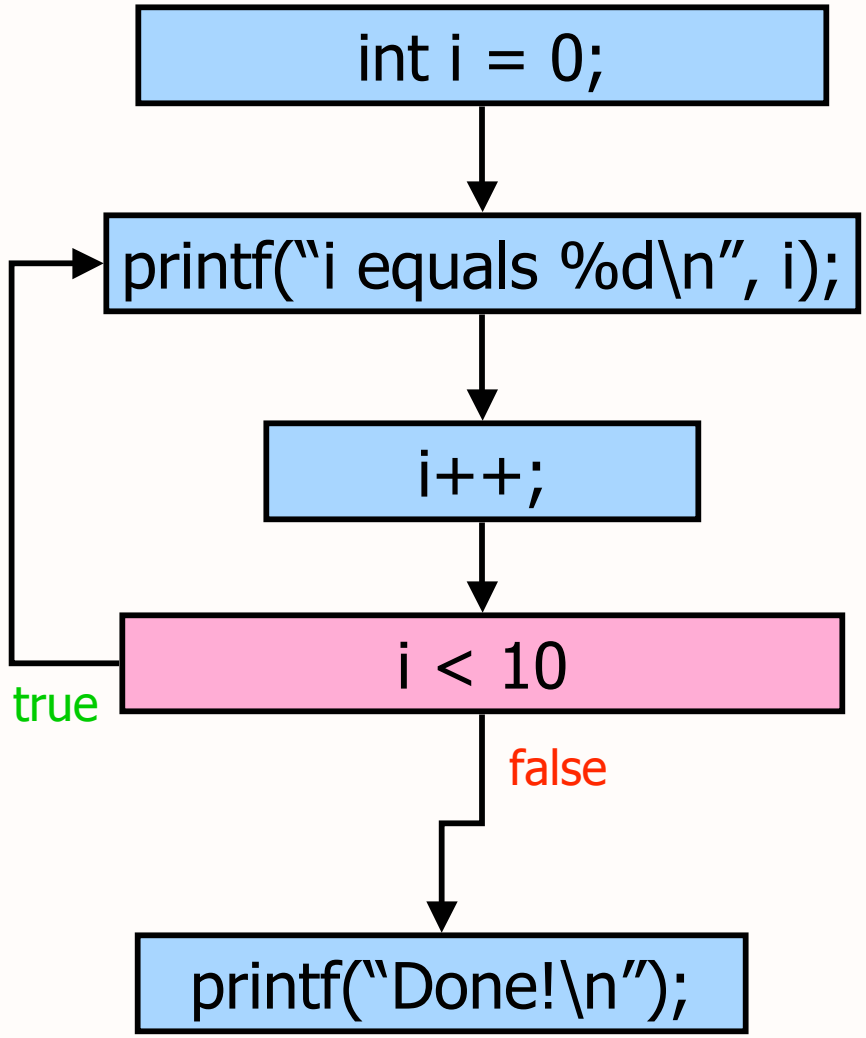
```
do {
    statements;
} while( condition );
```

Braces are optional if only one statement

- Statements are executed *at least once,* independent of condition

Do-While Loop

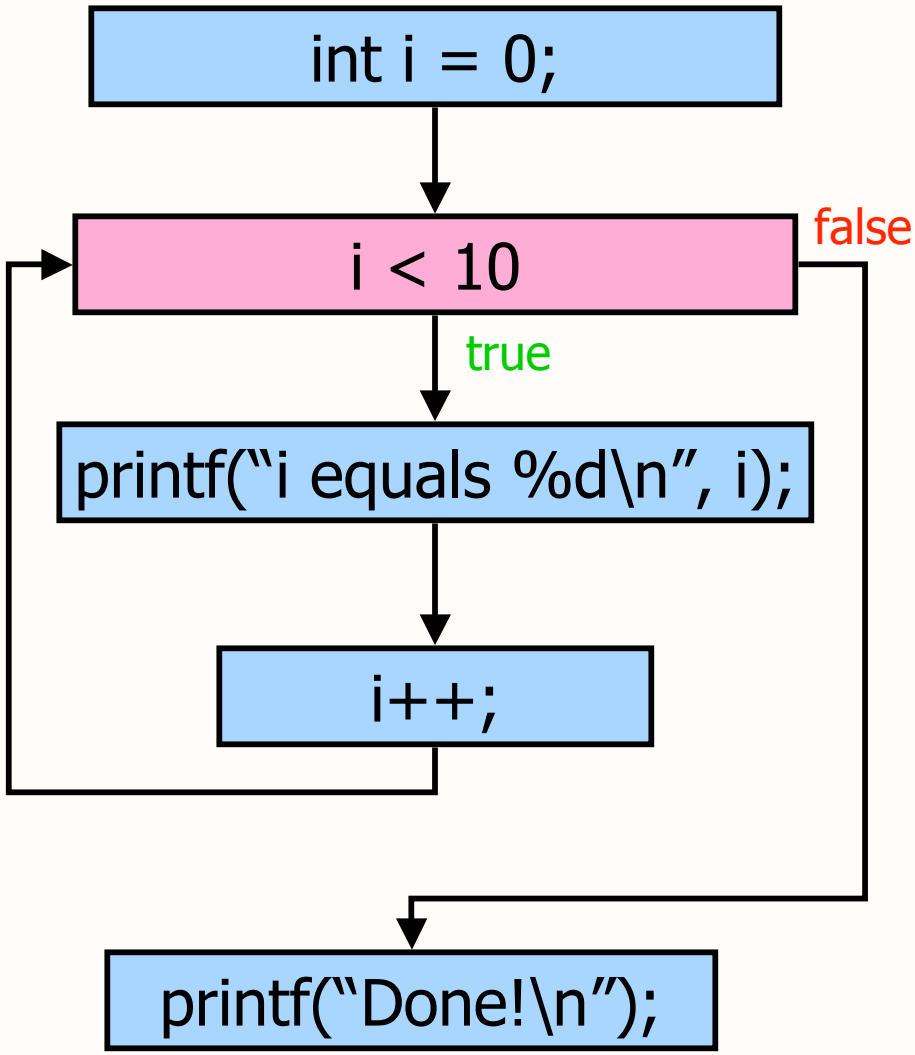
```
int i = 0;
do {
    printf("i equals %d\n", i);
    i++;
} while ( i < 10 );
printf("Done!\n");
```

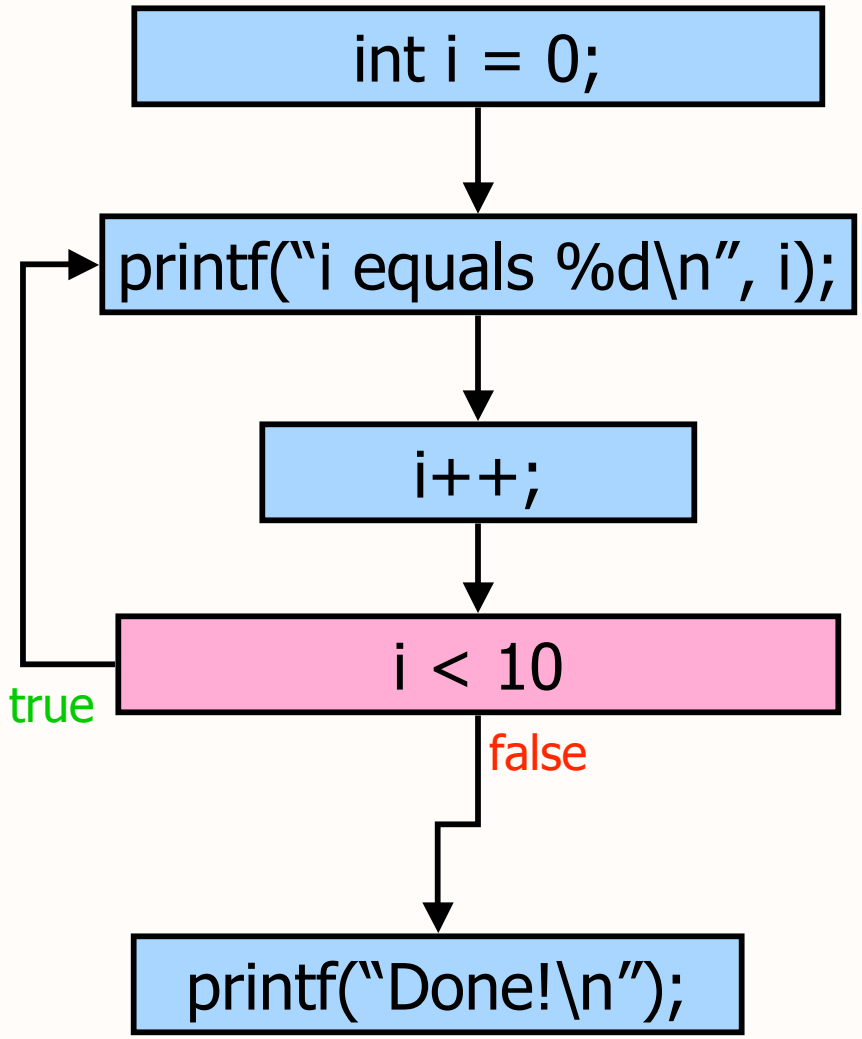


Questions:

- How many times will "i" get printed out?
- How many times is the condition evaluated?

While Loop vs. Do-While Loop





Loop Practice

- Change temperature converter user menu to use do-while loop

For Loop

- Good for when know how many times loop will execute
  - Repeat X times

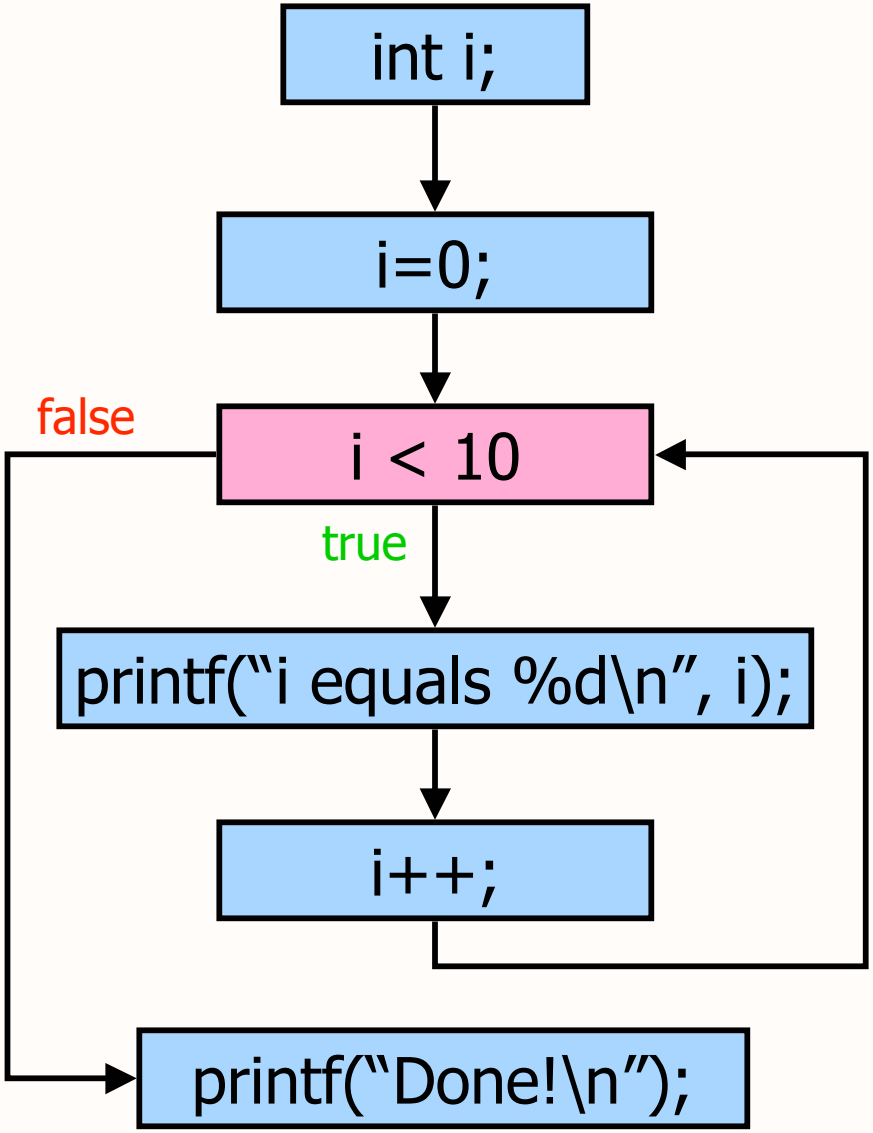
```
for( initialization; condition; execution_expr ) {
   statements;
}
```

Executed at end of each iteration

- Counter variable associated with the loop
  - Initialized, part of condition
  - Execution_expr: typically increments or decrements counter variable

For Loop

```
int i;
for( i=0; i < 10; i++ ) {
    printf("i equals %d\n", i);
}
printf("Done!\n");
```



Questions:

- How many times will "i" get printed out?
- How many times is the condition evaluated?

For Loop

- More condensed than while loop
- Harder to forget to increment or decrement counter

- Practice problem
  - Add 5 numbers, inputted by the user

Summary of Building Blocks

- Math library functions
  - pow, sqrt, ceil, sin, etc.
- Conditional statements
  - if, if-else, if-else-if, switch
- Loops
  - while, do-while, for

Practice Problems

- Extend compute average program
  - User input
  - Loop: Keep track of how many numbers inputted and average