QUIZ: What value is stored in a after this statement is executed? Why?

\[ a = 23/7; \]
5. To what value does the expression $10 \% 3$ evaluate?

6. To what value does the expression $5 + 3 \times 8 / 2 + 2$ evaluate?

7. Rewrite the expression in question 6, adding parentheses so that it evaluates to 16.
Lesson 4

Statements, Expressions, Operators
Statement = complete instruction that directs the computer to carry out a task

• Except for the preprocessor directives, C statements always end with a semicolon

```c
printf("Enter the height and the radius: ");
scanf("%u%u", &H, &R);
area = (float)H/2;

if (area > 42)
    printf("qwerty");
```

• Some statements extend over multiple lines (stay tuned!)
What is “whitespace”?

Any combination of:

• Space
• Horizontal tab \t
• Vertical tab \v
• Blank line (newline) \n
What is “vertical tab”?

• In the original ASCII code, it was a control character used to tell a printer to advance the paper vertically 6 lines.

• What we see in the MS Console window is the “male” symbol, as per IBM’s Code Page 437.

“The C compiler does not care about whitespace”
“The C compiler does not care about whitespace”

```c
void main() {
    char buffer[256];
}
```

- ... as long as the identifiers are still clearly separated
- The rule does not apply to the inside of strings

"Hello, World!" is not the same string as "Hello, W orld!"
How to split a long string on multiple lines

```c
printf("Hello, supercalifragilisticexpialidocious World!\n");
```

Typo on p.60 of text

**DON’T** forget to use a forward slash to continue a string of characters onto a second line.
How to create null statements

; ; ; ;
  fgets(buffer, 42, stdin);
 ;

They are sometimes during code development.
Block = Compound Statement

```
void main()
{
    char buffer[256];
    {
        printf("Name:\n");
        fgets(buffer, 42, stdin);
    }
    printf("there are %d characters and spaces\n", strlen(buffer));
}
```

Blocks can be nested within blocks (on any number of levels)
Expression = anything that evaluates to a number

Examples:
• Symbolic constants
• Literal constants
• Variables
Expressions can be nested on multiple levels, and they can include operators:

- Assignment oper.
- Math oper.
- Relational oper.
- Logical oper.
- `sizeof` oper.
Assignment operator \( = \)

- Usually only one assignment per line
- A few exceptions are considered “C programming style” (will mention them in the future)
- If multiple assignment in the same statement, the order of evaluation is right-to-left: \( a = b = c = 42; \)
Arithmetic operators

• Unary: increment, decrement, change-of-sign, plus

• Binary: add, subtract, divide, modulus (remainder), multiply

Typo on p.62 of text

C’s mathematical operators perform mathematical operations such as addition and subtraction. C has **two** unary mathematical operators and five binary mathematical operators.
Unary + is rarely used, but it exists for two reasons:
- symmetry with unary –
- promotion of shorter integers (char and short) to int

```c
printf("%d\n", sizeof( (char) 'x' ));
printf("%d\n", sizeof(+ (char) 'x' ));
printf("%d\n", sizeof( (long long) 'x' ));
printf("%d\n", sizeof(+ (long long) 'x' ));
```
Pre-increment vs. post-increment

There is no difference if the increment is the only operation in the statement:

```plaintext
a = 42;
++a;
```

In both cases, `a` has the same final value (43).
Pre-increment vs. post-increment

\[
\begin{align*}
a &= 42; \\
b &= a++; \\
a &= 42; \\
b &= ++a;
\end{align*}
\]

In both cases, \(a\) has the same final value (43), but \(b\) is different!
Pre-increment vs. post-increment

```
16:    printf("Count up!\n");
17:    printf("Post Pre\n");
18:    printf("%d %d\n", a++, ++b);
19:    printf("%d %d\n", a++, ++b);
20:    printf("%d %d\n", a++, ++b);
21:    printf("%d %d\n", a++, ++b);
22:    printf("%d %d\n", a++, ++b);
23:    printf("\nCurrent values of a and b:\n");
24:    printf("%d\n", a, b);
25:    printf("\nPost Pre\n");
26:    printf("Count down!\n");
27:    printf("Post Pre\n");
28:    printf("%d", a--, --b);
29:    printf("%d", a--, --b);
30:    printf("%d", a--, --b);
31:    printf("%d", a--, --b);
32:    printf("%d", a--, --b);
33:    printf("%d\n", a--, --b);
```

\text{a and b are both integers initialized with zero. Can you predict what the output will be?}
<table>
<thead>
<tr>
<th>Category</th>
<th>Operator</th>
<th>Associativity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Postfix</td>
<td>0 [], -&gt;, ., ++, -</td>
<td>Left to right</td>
</tr>
<tr>
<td>Unary</td>
<td>+, - !, ~, ++, - (type) * &amp; sizeof</td>
<td>Right to left</td>
</tr>
<tr>
<td>Multiplicative</td>
<td>* / %</td>
<td>Left to right</td>
</tr>
<tr>
<td>Additive</td>
<td>+, -</td>
<td>Left to right</td>
</tr>
<tr>
<td>Shift</td>
<td>&lt;&lt;, &gt;&gt;</td>
<td>Left to right</td>
</tr>
<tr>
<td>Relational</td>
<td>&lt;=, &gt;=</td>
<td>Left to right</td>
</tr>
<tr>
<td>Equality</td>
<td>==, !=</td>
<td>Left to right</td>
</tr>
<tr>
<td>Bitwise AND</td>
<td>&amp;</td>
<td>Left to right</td>
</tr>
<tr>
<td>Bitwise XOR</td>
<td>^</td>
<td>Left to right</td>
</tr>
<tr>
<td>Bitwise OR</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Logical AND</td>
<td>&amp;&amp;</td>
<td>Left to right</td>
</tr>
<tr>
<td>Logical OR</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Conditional</td>
<td>?:</td>
<td>Right to left</td>
</tr>
<tr>
<td>Assignment</td>
<td>=, +=, -=, *=, /=, %=, &gt;&gt;=, &lt;&lt;= &amp; = ^=</td>
<td>Right to left</td>
</tr>
<tr>
<td>Comma</td>
<td>,</td>
<td>Left to right</td>
</tr>
</tbody>
</table>
QUIZ: What values are stored in \texttt{a} and \texttt{b} after these statements are executed?

```c
int main(void) {
    int a;
    int b = 42;
    a = 2 * b++;
    printf("%d %d", a, b);
}
```
QUIZ: What values are stored in `a` and `b` after these statements are executed?

```c
int main(void) {
    int a;
    int b = 42;
    a = 2 * b++;
    printf("%d %d", a, b);  // 84 43
}
```
QUIZ: What values are stored in a and b after these statements are executed?

```c
int main(void) {
    int a;
    int b = 42;
    a = 2 * ++b;
    printf("%d %d", a, b);
}
```
QUIZ: What values are stored in a and b after these statements are executed?

```c
int main(void) {
    int a;
    int b = 42;
    a = 2 * ++b;
    printf("%d %d", a, b);  // 86 43
}
```
To do for next time:
Read pp.58-69 of text
QUIZ

Does the C compiler really “not care about whitespace”? Explain!
Is there any difference between these two fragments of code?

```
x = 42;
y = ++x;
```

```
x = 42;
y = x++;
```
QUIZ

What are the final values of x, y, and x?

\[
\begin{align*}
x &= y = 42; \\
y &= +=x/2; \\
z &= x++ + y--; \\
\end{align*}
\]
Boolean values: 1 (true) 0 (false)

The C language does not have a separate data type for Boolean – it uses integers instead!
### TABLE 4.4  C’s Relational Operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Symbol</th>
<th>Question Asked</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Equal</td>
<td>==</td>
<td>Is operand 1 equal to operand 2?</td>
<td>x == y</td>
</tr>
<tr>
<td>Greater than</td>
<td>&gt;</td>
<td>Is operand 1 greater than operand 2?</td>
<td>x &gt; y</td>
</tr>
<tr>
<td>Less than</td>
<td>&lt;</td>
<td>Is operand 1 less than operand 2?</td>
<td>x &lt; y</td>
</tr>
<tr>
<td>Greater than or equal to</td>
<td>&gt;=</td>
<td>Is operand 1 greater than or equal to operand 2?</td>
<td>x &gt;= y</td>
</tr>
<tr>
<td>Less than or equal to</td>
<td>&lt;=</td>
<td>Is operand 1 less than or equal to operand 2?</td>
<td>x &lt;= y</td>
</tr>
<tr>
<td>Not equal</td>
<td>!=</td>
<td>Is operand 1 not equal to operand 2?</td>
<td>x != y</td>
</tr>
</tbody>
</table>

### TABLE 4.5  Relational Operators in Use

<table>
<thead>
<tr>
<th>Expression</th>
<th>How It Reads</th>
<th>What It Evaluates To</th>
</tr>
</thead>
<tbody>
<tr>
<td>5 == 1</td>
<td>Is 5 equal to 1?</td>
<td>0 (false)</td>
</tr>
<tr>
<td>5 &gt; 1</td>
<td>Is 5 greater than 1?</td>
<td>1 (true)</td>
</tr>
<tr>
<td>5 != 1</td>
<td>Is 5 not equal to 1?</td>
<td>1 (true)</td>
</tr>
<tr>
<td>(5 + 10) == (3 * 5)</td>
<td>Is (5 + 10) equal to (3 * 5)?</td>
<td>1 (true)</td>
</tr>
</tbody>
</table>
QUIZ: What do these expressions evaluate to?

(7 == 5)
(5 > 4)
(3 != 2)
(6 >= 6)
(5 < 5)
QUIZ:
Assume that a is 2, b is 3 and c is 6. What do these expressions evaluate to?

(a == 5)
(a*b >= c)
(b+4 > a*c)
((b=2) == a)
```c
/* Demonstrates the evaluation of relational expressions */
#include <stdio.h>

int a;

int main()
{
    a = (5 == 5);  /* Evaluates to 1 */
    printf("\na = (5 == 5) \na = %d", a);

    a = (5 != 5);  /* Evaluates to 0 */
    printf("\na = (5 != 5) \na = %d", a);

    a = (12 == 12) + (5 != 1); /* Evaluates to 1 + 1 */
    printf("\na = (12 == 12) + (5 != 1) \na = %d\n", a);
    return 0;
}
```
Relational operators are used almost exclusively in conditions for `if` statements.
Relational operators are used almost exclusively as conditions for `if` statements

```c
19: age = CURRENTYEAR - birth_year;
20:  // Can check on voting age as well as drinking age
21:  if (age >= 18)
22:    printf("You can vote this year!\n");
23:  if (age <= 21)
24:    printf("It is illegal for you to drink alcohol!\n");
25:  return(0);
26: }
```
Listing 4.4  agechecker2.c: An if Statement with an else Clause

1: // Demonstrates if and else statements and some of C's relational operators
2: 
3: #define CURRENTYEAR 2013
4: #include <stdio.h>
5: 
6: int birth_year, age;
7: 
8: int main(void)
9: {
10:     printf("Enter the year you were born: ");
11:     scanf("%d", &birth_year);
12: 
13:     // Two tests to calculate whether the user was a leap year birth
14:     if (birth_year % 4 == 0)
15:         printf("You were born in a leap year!\n");
16:     else
17:         printf("You were not born in a leap year!\n");
#include <stdio.h>

int main ()
{
    /* local variable definition */
    int a = 100;

    /* check the boolean condition */
    if( a == 10 )
    {
        /* if condition is true then print the following */
        printf("Value of a is 10\n");
    }
    else if( a == 20 )
    {
        /* if else if condition is true */
        printf("Value of a is 20\n");
    }
    else if( a == 30 )
    {
        /* if else if condition is true */
        printf("Value of a is 30\n");
    }
    else
    {
        /* if none of the conditions is true */
        printf("None of the values is matching\n");
    }
    printf("Exact value of a is: %d\n", a);
    return 0;
}
### Table 4.7 C’s Logical Operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Symbol</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>AND</td>
<td>&amp;&amp;</td>
<td><code>exp1 &amp;&amp; exp2</code></td>
</tr>
<tr>
<td>OR</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NOT</td>
<td>!</td>
<td><code>!exp1</code></td>
</tr>
</tbody>
</table>

**Truth tables (not in our text):**

| P | Q | P && Q | P || Q | !P |
|---|---|--------|--------|----|
| T | T |  T     |  T     |  F |
| T | F |  F     |  T     |  F |
| F | T |  F     |  T     |  T |
| F | F |  F     |  F     |  T |

Expression is true if and only if both operands are true.

Expression is true if and only if either operand is true (incl. both!).

### TABLE 4.7  C’s Logical Operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Symbol</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>AND</td>
<td>&amp;&amp;</td>
<td><code>exp1 &amp;&amp; exp2</code></td>
</tr>
<tr>
<td>OR</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NOT</td>
<td>!</td>
<td><code>!exp1</code></td>
</tr>
</tbody>
</table>

Expression is true if and only if both operands are true.

Expression is true if and only if either operand is true (incl. both!).

### TABLE 4.9  Code Examples of C’s Logical Operators

<table>
<thead>
<tr>
<th>Expression</th>
<th>What It Evaluates To</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>(5 == 5) &amp;&amp; (6 != 2)</code></td>
<td>True (1), because both operands are true</td>
</tr>
<tr>
<td>`(5 &gt; 1)</td>
<td></td>
</tr>
<tr>
<td><code>(2 == 1) &amp;&amp; (5 == 5)</code></td>
<td>False (0), because one operand is false</td>
</tr>
<tr>
<td><code>!(5 == 4)</code></td>
<td>True (1), because the operand is false</td>
</tr>
</tbody>
</table>
int a = 5, b = 6, c = 5, d = 1;
int x;

int main( void )
{
    /* Evaluate the expression without parentheses */

    x = a < b || a < c && c < d;
    printf("\nWithout parentheses the expression evaluates as %d", x)

    /* Evaluate the expression with parentheses */

    x = (a < b || a < c) && c < d;
    printf("\nWith parentheses the expression evaluates as %d\n", x);
}
QUIZ

What is the difference between relational and logical operators?
What is the difference between **relational** and **logical** operators?

- Relational operator: `>`
  - Evaluates to: `false`

- Logical operator: `||` (logical OR)
  - Evaluates to: `true`
QUIZ

Is \(-42\) true or false?
QUIZ

What is a more economical way to write

```c
if (a != 0) ?
```
QUIZ: The integer variable `score` contains the score a student got on an exam (0-100). Write a decision statement (`if`) to print the corresponding `grade` (F-A).
QUIZ: What is the final value of \( a \)?

\[
a = !(1<=2) \quad || \quad 3>2 \quad && \quad 3+4>7;
\]

Hint: Use the precedence table!
Extra-credit QUIZ
The only ternary operator: conditional op.

```
x = y ? 1 : 100;
```

```
z = (x > y) ? x : y;
```

```
if (x > y)
z = x;
else
z = y;
```
QUIZ: Rewrite using conditional op.

```c
if (birth_year % 4 == 0)
    printf("You were born in a leap year!\n");
else
    printf("You were not born in a leap year!\n");
```
<table>
<thead>
<tr>
<th>When You Write This</th>
<th>It Is Equivalent to This</th>
</tr>
</thead>
<tbody>
<tr>
<td>x *= y</td>
<td>x = x * y</td>
</tr>
<tr>
<td>y -= z + 1</td>
<td>y = y - z + 1</td>
</tr>
<tr>
<td>a /= b</td>
<td>a = a / b</td>
</tr>
<tr>
<td>x += y / 8</td>
<td>x = x + y / 8</td>
</tr>
<tr>
<td>y %= 3</td>
<td>y = y % 3</td>
</tr>
</tbody>
</table>
A **comma-separated** expression is evaluated left-to-right

```c
int a = 2, b = 3;
x = (a++, b++);
```
A **comma-separated** expression is evaluated left-to-right

```c
int a = 2, b = 3;

x = (a++ , --b, a += b);
```
To do in notebook for next time:

Read the entire Lesson 4
Solve all quizzes we solved in class
Homework for Chapters 3 and 4:  Due Wed, Feb.3
• Ch.3 – Exercise 4 (no code)
• Ch.4 – Exercises 7 (no code), 9 (code), 10 (code)

For each coding exercise capture a screenshot of both code and output.

Turn in hardcopy at beginning of class on due date!