Combined Assignments, Relational Operators, and the If Statement

Multiple Assignments (3.7)

- C++ allows statements such as:

```
a = b = c = d = 45;
```

- Why do you think that is?
- What is the associativity of the assignment operator?

Combined Assignments

- The same variable can be used on the left hand side of the assignment and on the right hand side

```
notes = notes / 20;
note = notes % 20;
```

- These are common in programming, so the two operators can be combined as follows:

```
notes /= 20;
note %= 20;
```

Examples of Combined Assignments

Operator	*Example Usage*	*Equivalent To*
+=	`x += 5;`	`x = x + 5;`
-=	`y -= 2;`	`y = y – 2;`
*=	`z *= 10;`	`z = z * 10;`
/=	`a /= b;`	`a = a / b;`
%=	`c %= 3;`	`c = c % 3;`

8.1 Combined Assignments

- Combined assignments can be combined with arithmetic operators

```
y -= a * 2;
a /= b + c;
C %= d – 3;
```

- What is the long form of these statements?

8.2 What is the Output?

```
int unus, duo, tres;

unus = duo = tres = 5;
unus += 4;
duo *= 2;
tres -= 4;
unus /= 3;
duo += tres;
cout << unus << endl;
cout << duo << endl;
cout << tres << endl;
```

getline (3.9)

- What happens when the user types their first and last name for the following code segment?

```
string name;
cout << "Enter your name: ";
cin >> name;
```


getline

- **`cin`** passes over and ignores leading whitespaces, but will stop reading once it gets to the first whitespace character after the string
- Solution?
  - Use **`getline`** function

getline

```
string name;
cout << "Enter your name: ";
getline(cin, name);
```

getline

- Syntax for getline

```
getline(cin, inputLine);
```

- Where
  - **cin** is the input stream
  - **inputLine** is the variable where the string will be stored

cin.get()

- Used to read one character from the keyboard at a time
- Also reads new lines, spaces, and tabs as a character
  - '\n': new line
  - '\t': tab
  - ' ': space

Example

```
char ch;
cout << "This program has paused.";
cout << "Press Enter to continue.";
cin.get(ch);
cout << "Thank you!" << endl;
```

Relational Operators (4.1)

- So far, we can Input, Output and Calculate
- How can we explore relationships between data?
  - Is your grade greater than 90%?
  - Is it hotter or colder today than yesterday?
  - Do I have enough US dollars to get 100 Euros?

Relational Operators, Explained!

Operator	*Meaning*
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to
==	Equal to
!=	Not equal to

- All are binary operators
- Left to right associativity

Relational Expression

- An expression is a statement that has value
- Relational expression: an expression that uses a Relational Operator
  - its value is a Boolean value (True or False)

```
int x = 9, y = 42;
```

  - `x > y`
  - `y == x // y = x; is the assignment operator`
  - `x <= (x * y + 99)`

Precedence (page 1125)

Precedence Operators (Highest to Lowest)					
(unary negation) -					
*	/	%			
+	-				
>	>=	<	<=		
==	!=				
=	+=	-=	*=	/=	%=

Arithmetic Operators

Relational Operators

Assignment Operators

8.3 Practice

- What is the value of the following Relational Expressions?

```
int x = 99, y = 42;
```

  - `x > y`
  - `y <= x`
  - `y != x`
  - `x == (x + 1)`
  - `y == y + 1`

Boolean value (True or False)

- How does the computer represent True and False?
- New data type: `bool`

```
bool tValue = true;  // 1
bool fValue = false; // 0
```

8.4 Practice

```
bool value;
int x = 5, y = 10;
value = x > y; // value = ??
value = x == y; // value = ??
value = x == y - 5; // value = ??

// what does this output look like?
cout << "Value is: " << value;
```


The `if` Statement (4.2)

- We execute each statement in our program in order
- What `if` we only want to execute a statement sometimes?
- The `if` Statement!

```
int x=5, y=10;

if(x > y)
{
   // do stuff
}
```


Formally defined

```
if(expression)
{
   statement 1;
   statement 2;
   . . .
   statement n;
}
```

Just like a **function**, start at the top and execute in order to the bottom

- What is an expression?

8.5 Practice

```
int x = 5, y = 10;
bool value = x > y;

if (value)
{
  cout << "value is True" << endl;
}
if (x < y)
{
  cout << x << " < " << y;
  cout << " is true" << endl;
}
```


Coding Standards

```
if(expression)
{
   statement 1;
}
```

```
if(expression)
   statement 1;
```

If you only have ONE statement in the body of the if, the { } are optional in C++.

For this class, the { } must **ALWAYS** be used. Not using { } will result In a loss of style points.

The { } must also be on their own line.

Why?

8.6 More on Truth

- Expressions that evaluate to non-zero are considered `true`

```
int x = 5, y = 0;
if (x + y)
{ // This will be executed
  cout << "x + y is True" << endl;
}
if (y)
{ // This will NOT be executed
  cout << "y is True" << endl;
}
```