Random useful stuff

STAY STRONG!

WEEKEND IS COMING SOON

Funnymemes.com.au
Announcements

HW1 is posted on schedule, due next Thursday at 6pm (pretty easy)

If you have trouble compiling on cselabs, type this in a terminal:

```bash
module rm soft/gcc/4.5.2
module initrm soft/gcc
```
Input and output

C:\> If you're happy and you know it, syntax error!
Syntax error

C:\> If you're happy and you know it, syntax error!
Syntax error

C:\> If you're happy and you know it, and you really want to show it, if you're happy and you know it, syntax error!
Syntax error

C:\> _
Strings and input

char can only hold a single letter/number, but one way to hold multiple is a string

```cpp
string str;
 cin >> str;
```

The above will only pull one word, to get all words (until enter key) use:

```cpp
getline(cin, str);  (See: stringInput.cpp)
```
More Output

When showing doubles with cout, you can change how they are shown.

For example, to show a number as dollars and cents, you would type (before cout):

```
cout.setf(ios::fixed);
cout.setf(ios::showpoint);
cout.precision(2);
```
More Output

There are two ways to get output to move down a line: `endl` and `"\n"

```cpp
    cout << endl;
```

... is the same as...

```cpp
    cout << "\n"
```

I will use both when coding
Madlibs

(see: madlibs.cpp)
bool

bool - either true or false

We will use the following today:

>  (greater than), e.g. 7 > 2.5 is true
==  (equals), e.g. 5 == 4 is false
<=  (less than or eq), e.g. 1 <= 1 is true
if statement

Code inside an **if** statement is only run if the condition is true.

Need parenthesis (no semi-colon)

```cpp
if (guess == random0to9)
{
    cout << "Correct, here is a cookie!\n";
}
```

Indent

(See: numberGuessing.cpp)
if/else statement

Immediately after an if statement, you can make an else statement

If the “if statement” does not run, then the else statement will

If you do not surround your code with braces, only one line will be in the if (and/or else) statement
Complex expressions

If statements for when \( x \)...

... is between 10 and 20 (inclusive)

\[
\text{if}(10 \leq x \ \&\& \ x \leq 20)
\]

Cannot say: 10 \( \leq \) \( x \) \( \leq \) 20 (why?)

... is a vowel (\( x \) is type \texttt{char})

\[
\text{if}( \ x == 'a' \ |\ | \ x == 'e' \ |\ | \ x == 'i' \ |\ | \ x == 'o' \ |\ | \ x == 'u')
\]
Double trouble!

(See: doubleCompare.cpp)
Double trouble!

When comparing doubles, you should use check to see if relative error is small:

$$\text{fabs}((x-y)/x) < 10\text{E-10}$$

(double has about 16 digits of accuracy so you could go to 10E-15 if you want)

For comparing Strings, use: (0 if same)

```
string1.compare(string2)
```
Short-circuit evaluation is when you have a complex bool expression (&& or ||) but you don't need to compute all parts.

```cpp
if(false && 7/0 == 2) {
    cout << "Will I crash?\n";
}
```

If this is false, then it will not check next (See: shortCircuit.cpp)
Short-circuit evaluation

Simple cases of short-circuit:
When you have a bunch of ORs
   if( expression || exp || exp || exp )
Once it finds any true expression, if statement will be true

When you have a bunch of ANDs
   if( expression && exp && exp && exp )
Once it finds any false expression, if statement will be false
Loop

Often we want to do a (similar) command more than once

Computer programmers call this code a loop

Loops are quite powerful and are very commonly used
A while loop tests a **bool** expression and will run until that expression is **false**.

```cpp
while (i < 10) {
    // looped code
    // variable i should change in here
}
```

(See: whileLoop.cpp)
while loop

The `bool` expression is tested when first entering the while loop.

And!

When the end of the loop code is reached (the `}` to close the loop)

```cpp
int i = 0;
while (i < 5) {
    cout << "Looping, i = " << i << "\n";
    i++;
}
cout << "Outside the loop, i = " << i << "\n";
```
while loop

3 parts to any (good) loop:

- Test variable initialized
  
i = 0;

- bool expression
  
while (i < 10)

- Test variable updated inside loop
  
i++;
A do-while loop is similar to a normal while loop, except the `bool` expression is only tested at the end of the loop (not at the start).

```cpp
cout << "How many times do you want to run the loop?\n";  
cin >> i;  // what happens if i is less than 1?  
    do {  
        cout << "Looping, i = " << i << "\n";  
        i--;  
    } while (i > 0);  // Note semicolon!  
cout << "Outside the loop, i = " << i << "\n";  
```

(See: doWhile.cpp)
do-while loop

Q: Why would I ever want a do-while loop?

A: When the first time the variable is set is inside the loop.

You can initialize the variable correctly and use a normal while loop, but this makes the logic harder.
Semi-colons

When to put semi-colons?

You put semi colons after every statement, except if you (should) start a block

Blocks should start after functions, if-statements and loops, thus these should not have semi colons after them