Before the lecture begins …

- Make sure you have Homework 1/1b turned in.
- Review the Lecture 3 slides.

Intro to C++

Lecture 3

while loops, for loops, casting, order of operations

Loops

- A ***loop*** occurs when certain portions of code are executed more than once in succession.
- Loops are important in computing to save time writing code, save hard drive space, and to manipulate data that has a size undeterminable at compilation time.

while loop

- A loop you are already familiar with is the ***while loop***, which says, 'While a condition is true, execute the commands in braces'.
- In the graphics framework, a ***while loop*** is used to render graphics *while* the escape key is not pressed.
- Notice that just like with if statements, the ***while loop*** tests whether the condition in parentheses is *true* before making a decision.

while Loop Example

```
int done = 1;
while (!done)
{
     // Commands here
}
```

- Will the code inside the ***while loop*** ever be executed in this scenario?
- Remember that in C/C++, *true* is any non-zero number and that the '!' symbol is read as 'not'.
- Thus, the commands will *not* be executed because first, done is evaluated to be *true* (it is 0). Then, due to the '!', its value is negated to *false* which does not satisfy the loop's 'while the condition is *true…'*

do-while Loop

- Another related type of loop is the ***do-while loop***. It is exactly the same as the ***while loop*** except that the conditional expression is evaluated at the *end* of the loop instead of at the beginning, guaranteeing that the code in braces is executed at least once.

do-while Loop Example

```
int done = 1;
do
{
     // Commands here
} while (!done);
```

- Will the code inside the ***do-while loop*** ever be executed in this scenario?
- Yes! Although the loop will terminate once the expression !done is evaluated, the evaluation does not occur until the *bottom* of the loop, meaning that the commands will be executed once.

When to Use while/do-while Loops

- You have a single variable to test for each iteration.
- No counters are dependent on the loop iteration.
- Use while when you only want to execute the loop commands if the condition is true (e.g. set off an alarm only while the door is touched).
- Use do-while when you want to execute the inside at least once (e.g. when drawing a menu and asking the user which choice he wants, continually draw it until he enters a valid choice).

for Loops

- The ***for loop*** usually is used when a loop must execute a predefined number of times.
- This loop has a counter that increments every loop iteration so you can also easily tell which iteration you're at and use that in your calculations. The loop also allows you to initialize the counter to a set value when the loop is first begun.

for Loop Example

```
int i = 1;
for(i=0; i<10; i++)
{
     // Commands to be executed
}
```

equivalent

```
int i = 0;
while(i<10)
{
     // Commands to be executed

     i++;
}
```

1. i is *initialized* to zero.
2. If i is less than ten, the commands are executed.
3. i is incremented by one.
4. Go to 2.

for Loop Example

```
int linePos = 1;
for(linePos=0; linePos<10; linePos+=2)
{
     itcc->SetPixel(xPos+linePos, yPos, RGB32(255,255,255));
}
```

Here, we draw a dotted horizontal line, beginning at the coordinates (xPos, yPos) (note that linePos = 0) and ending at the coordinates (xPos + 9, yPos) (linePos = 9), for a total of ten pixels.

Casting

Problem:

```
long myScore = 100;
unsigned long totalScore = 0;

myScore = totalScore;
```

- What's the problem here? totalScore can potentially be larger than myScore, resulting in data loss!
- But what if we know for *sure* that totalScore will be able to fit inside myScore?
- We can ***cast*** the variable – this is a technique used to tell the compiler that we realize that the fit is not perfect, and that we understand the possible consequences, but that we want to do this anyway.

Solution:

```
long myScore = 100;
unsigned long totalScore = 0;

myScore = (long)totalScore;
```

- To perform a ***cast***, merely put the data type the variable will end up as in parentheses in front of the entire expression you want converted.

Casting

- There are many times in C/C++ where ***casting*** is necessary, although it should be attempted as infrequently as possible.
- Why? Not only might this actually lead to hard-to-find errors when converting, but it is usually a very slow process (*especially* when converting between floating point numbers and integers!).

Order of Operations

- As taught in mathematics courses, when lots of mathematics symbols are all jumbled together in an expression, don't some take precedence over others?
- Look this up on the Internet for more detailed charts, but see the next slide for the order of operations for a few operators we've investigated so far!

Order of Operations

- **Parentheses** are *always* evaluated first!
- Prefix/postfix **increment/decrement** operators are evaluated.
- **Multiplications** and **divisions** are evaluated from left to right.
- **Additions** and **subtractions** are evaluated from left to right.
- **Bit shifts** are the last mathematical operators to be evaluated.

Order of Operations Example

color = red << 16 + green / 4<< 8 + blue;

- Will this work correctly? NO! First, the division is evaluated – green/4 (as we want). However, bit shifts are performed AFTER addition, so we need to place parentheses around the bit shifts:

color = (red << 16) + ((green / 4) << 8) + blue;

- It's safest to use parentheses when you're not sure – better safe than sorry, because these errors are tough to catch!

TODO

- Download ITCC_HW2.zip from the site (will not be available until Wednesday, July 7) http://www.pclx.com/itcc/, and complete the homework exercises, emailing them (FOR THIS WEEK ONLY) to itcc_teachers@pclx.com. Please do not resubmit solutions, even if they are revised. All homework must be submitted by 6:00am PST Monday, July 12.
- Some illustrative examples of the topics in this lecture are given in ITCC_HW2.zip.
- If you still have problems compiling the framework, please make sure to get in contact with us IMMEDIATELY!
- Look over the slides for the third lecture before Monday, July 12.
- If you finish with the homework, experiment!