Announcements

• Lab 2 – Due Friday
Review

- Write a program to ask the user to enter two integers representing hours then minutes. Output the equivalent number of seconds.

- To get started...
  - Go to [http://bytes.usc.edu/cs103/in-class-exercises](http://bytes.usc.edu/cs103/in-class-exercises)
    - printseconds
  - We've started the program for you...look at the
    - General template for a program with the #includes, using namespace std; and int main() function which returns 0
  - We've declared variables where you can store the input and computation results
  - Now you add code to
    - Get input from the user
    - And compute the answer and place it in the 'sec' variable
If..else statements

MODULE 5: CONDITIONAL STATEMENTS
Comparison Operators

- Control structures like if, while, and for require conditions to determine what code should execute.
- To perform comparison of variables, constants, or expressions in C/C++ we can use the basic 6 comparison operators.

<table>
<thead>
<tr>
<th>Operator(s)</th>
<th>Meaning</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>==</td>
<td>Equality</td>
<td>if(x == y)</td>
</tr>
<tr>
<td>!=</td>
<td>Inequality</td>
<td>if(x != 7)</td>
</tr>
<tr>
<td>&lt;</td>
<td>Less-than</td>
<td>if(x &lt; 0)</td>
</tr>
<tr>
<td>&gt;</td>
<td>Greater-than</td>
<td>if(y &gt; x)</td>
</tr>
<tr>
<td>&lt;=</td>
<td>Less-than OR equal to</td>
<td>if(x &lt;= -3)</td>
</tr>
<tr>
<td>&gt;=</td>
<td>Greater-than OR equal to</td>
<td>if(y &gt;= 2)</td>
</tr>
</tbody>
</table>
Logical AND, OR, NOT

- Often want to combine several conditions to make a decision
- Logical AND => $x > 0 \land y > 0$
- Logical OR => $x = 1 \lor x = 2$
- Logical NOT => $!(x < 0)$
- Precedence (order of ops.) => $!$ then $\land$ then $\lor$
  - $!(cond1) \lor (cond2 \land !(cond3))$

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>AND</th>
</tr>
</thead>
<tbody>
<tr>
<td>False</td>
<td>False</td>
<td>False</td>
</tr>
<tr>
<td>False</td>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td>True</td>
<td>False</td>
<td>False</td>
</tr>
<tr>
<td>True</td>
<td>True</td>
<td>True</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>OR</th>
</tr>
</thead>
<tbody>
<tr>
<td>False</td>
<td>False</td>
<td>False</td>
</tr>
<tr>
<td>False</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>True</td>
<td>False</td>
<td>True</td>
</tr>
<tr>
<td>True</td>
<td>True</td>
<td>True</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>A</th>
<th>NOT</th>
</tr>
</thead>
<tbody>
<tr>
<td>False</td>
<td>True</td>
</tr>
<tr>
<td>True</td>
<td>False</td>
</tr>
</tbody>
</table>
Exercise

Which of the following is NOT a condition to check if the integer \( x \) is in the range \([-1 \text{ to } 5]\)

- \( x \geq -1 \land x \leq 5 \)
- \(-1 \leq x \leq 5\)
- \(! (x < -1 \lor x > 5)\)
- \(x > -2 \land x < 6\)
bools, ints, and Conditions

• Loops & conditional statements require a **condition** to be evaluated resulting in a **true** or **false** result.

• In C/C++...
  – 0 means **false** / Non-Zero means **true**
  – **bool** type available in C++ => ‘**true**’ and ‘**false**’ keywords can be used but internally
    • **true** = non-zero (usually 1) and
    • **false** = 0

• Any place a condition would be used a bool or int type can be used and will be interpreted as bool

```cpp
int x = 100;
if(x)
    { x--; }

bool done = false;
while( ! done )
    { cin >> done; }

int x=100, y=3, z=0;
if( !x || (y && !z) )
    { /* code */ }
```
Conditions and DeMorgans

• Write a condition that eats a sandwich if it has neither tomato nor lettuce
  – if ( !tomato && !lettuce) { eat_sandwich(); }
  – if ( !(tomato || lettuce) ) { eat_sandwich(); }

• DeMorgan's theorem says there is always two ways to express a logic condition
  – !a && !b ⇔ !(a || b)
  – !a || !b ⇔ !(a && b)

• More details in EE 109 and CS 170
If..Else Flow Chart

```
if (condition1)
{
    // executed if condition1 is true
}
else
{
    // executed if condition1 
    // above is false
}
// following statements
```
If...Else If...Else

- Use to execute only certain portions of code
- `else if` is *optional*
  - Can have any number of `else if` statements
- `else` is *optional*
- `{ ... }` indicate code associated with the if, else if, else block

```java
if (condition1)
{
    // executed if condition1 is true
}
else if (condition2)
{
    // executed if condition2 is true
    // but condition1 was false
}
else if (condition3)
{
    // executed if condition3 is true
    // but condition1 and condition2 were false
}
else
{
    // executed if neither condition above is true
}
```
**else if**

```java
if (condition1)
{
    // executed if condition1 is True
}
else if (condition2)
{
    // executed if condition2 is True
    // but condition1 was False
}
else
{
    // executed if neither condition
    // above is True
}
```

**These 2 are equivalent**

```java
if (condition1)
{
    // executed if condition1 is True
}
else
{
    if (condition2){
        // executed if condition2 is True
        // but condition1 was False
    }
    else
    {
        // executed if neither condition
        // above is True
    }
}
```
Single Statement Bodies

• **The Rule:** Place code for an if, else if, or else construct in curly braces `{ ... }`

• **The Exception:**
  – An if or else construct with a single statement body does not require `{ ... }
  – Another if counts as a single statement

• **Prefer `{ ... }** even in single statement bodies so that editing later does not introduce bugs

```plaintext
if (x == 5)
    y += 2;
else
    y -= 3;

if (x == 5)
    y += 2;
else
    if(x < 5)
        y = 6;
    else
        y = 0;
```
PROBLEM SOLVING IDIOMS
Rule/Exception Idiom

• **Name**: Rule/Exception
• **Description**: Perform a default action and then use an ‘if’ to correct for exceptional cases
• **Structure**: Default action code followed by if statement with code to correct the exceptional case
• **Example(s)**: 
  – Shipping for "members"

```cpp
// Default action
if( /* Exceptional Case */ )
{
    // Code to apply to
    // exceptional case
}

bool primeMember = /* set somehow */;

double shippingFee = 7.99;
if( primeMember == true )
{
    shippingFee = 0;
}
```
Look-up Table Idiom

• **Name**: Look-up Table (Parallel cases)
  – A table can describe the mapping of input to output

• **Description**: Break input into **mutually exclusive** cases, taking some action or producing some output in each case

• **Structure**: Single level
  'if..else if..else' statement

<table>
<thead>
<tr>
<th>Score (input)</th>
<th>Grade (output)</th>
</tr>
</thead>
<tbody>
<tr>
<td>&gt; 90</td>
<td>A</td>
</tr>
<tr>
<td>80-89</td>
<td>B</td>
</tr>
<tr>
<td>70-79</td>
<td>C</td>
</tr>
<tr>
<td>55-69</td>
<td>D</td>
</tr>
<tr>
<td>&lt; 55</td>
<td>F</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Weather</th>
<th>Dress</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hot</td>
<td>T-shirt</td>
</tr>
<tr>
<td>Mild</td>
<td>Long Sleeves</td>
</tr>
<tr>
<td>Cold</td>
<td>Sweater</td>
</tr>
</tbody>
</table>

```c
if( /* Condition 1 */ ) {
    // Case 1 code
}
else if( /* Condition 2 */ ) {
    // Case 2 code
}
else if( /* Condition 3 */ ) {
    // Case 3 code
}
else { /* Default */
    // Default code
}
```

```c
if( weather == "hot" ) {
    clothing = "t-shirt";
}
else if( weather == "mild" ) {
    clothing = "long sleeves";
}
else { /* Default */
    clothing = "sweater";
}
```
**Decision Tree (Subcase) Idiom**

- **Name**: Decision Tree (Subcase)
- **Description**: The result of one condition determines which condition (subcase) to check next
- **Structure**: Nested 'if' statements

```java
if( /* Condition 1 */ )
{
    // Case 1 code
    if( /* Subcondition 1a */ ) {
        // Subcase 1a code
    }
    else {
        // Subcase 1b code
    }
}
else if( /* Condition 2 */ )
{
    // Case 2 code
    if( /* Subcondition 2a */ ) {
        // Subcase 2a code
    }
}
```

---

**Customer Service Call Menu**

- **Top-level Cases**: Account Issues, Hours of Operation
- **Sub-Cases**: Balance, Cancel
Exercises

• Conditionals In-Class Exercises
  – discount
  – weekday
  – nth
The Right Style

- Is there a difference between the following two code snippets?
- Both are equivalent but
  - Two if statements implies both can execute
  - An if..else implies a mutually exclusive relationship where only 1 can execute
- For mutually exclusive cases, use if..else for clarity sake

```cpp
int x;
cin >> x;
if( x >= 0 ) { cout << "Positive"; } 
if( x < 0  ) { cout << "Negative"; } 

int x;
cin >> x;
if( x >= 0 ) { cout << "Positive"; } 
else { cout << "Negative"; }
```
Find the bug

• What's the problem with this code...

• Common mistake is to use assignment '=' rather than equality comparison '==' operator

• Assignment puts 1 into x and then uses that value of x as the "condition"
  – 1 = true so we will always execute the if portion

```cpp
// What’s the problem below
int x;
cin >> x;
if (x = 1)
    { cout << "x is 1" << endl; }
else
    { cout << "x is not 1" << endl; }
```

```cpp
// What’s the problem below
int x;
cin >> x;
if (x == 1)
    { cout << "x is 1" << endl; }
else
    { cout << "x is not 1" << endl; }
```
Switch (Study on own)

- Again used to execute only certain blocks of code
- **Cases must be a constant**
- Best used to select an action when an expression could be 1 of a set of constant values
- `{ ... } around entire set of cases and not individual case
- Computer will execute code until a break statement is encountered
  - Allows multiple cases to be combined
- Default statement is like an else statement

```java
switch(expr) // expr must eval to an int
{
    case 0:
        // code executed when expr == 0
        break;
    case 1:
        // code executed when expr == 1
        break;
    case 2:
    case 3:
    case 4:
        // code executed when expr is 2, 3, or 4
        break;
    default:
        // code executed when no other case is executed
        break;
}
```
Switch (Study on own)

- What if a break is forgotten?
  - All code underneath will be executed until another break is encountered

```java
switch(expr) // expr must eval to an int {
    case 0:
        // code executed when expr == 0
        break;
    case 1:
        // code executed when expr == 1
        // what if break was commented
        // break;
    case 2:
    case 3:
    case 4:
        // code executed when expr is
        // 3, 4 or 5
        break;
    default:
        // code executed when no other
        // case is executed
        break;
}
```
? Operator

• A simple if..else statement can be expressed with the ? operator
  – `int x = (y > z) ? 2 : 1;`
  – Same as:
    ```
    if(y > z) x = 2;
    else x = 1;
    ```
• Syntax: `(condition) ? expr_if_true : expr_if_false;`
• Meaning: the expression will result/return `expr_if_true` if `condition` evaluates to true or `expr_if_false` if `condition` evaluates to false
Performing repetitive operations

MODULE 6: LOOPS (ITERATIVE STATEMENTS)
Need for Repetition

• We often want to repeat a task but do so in a concise way
  – Print out all numbers 1-100
  – Keep taking turns until a game is over
  • Imagine the game of 'war'...it never ends!!
• We could try to achieve these without loops, but...

```cpp
#include <iostream>
using namespace std;

int main()
{
    cout << 1 << endl;
    cout << 2 << endl;
    ...
    cout << 100 << endl;
    return 0;
}
```

```cpp
#include <iostream>
using namespace std;

int main()
{
    bool gameOver;
    gameOver = take_turn();
    if( ! gameOver ){
        gameOver = take_turn();
        if( ! gameOver ) {
            ...
        }
    }
}
```

Assume this performs code to "take a turn" and then produces a true/false result indicating if the game is over.
4 Necessary Parts of a Loop

- Loops involve writing a task to be repeated
- Regardless of that task, there must be **4 parts** to make a loop work
- **Initialization**
  - Initialization of the variable(s) that will control how many iterations (repetitions) the loop will execute
- **Condition**
  - Condition to decide whether to repeat the task or stop the loop
- **Body**
  - Code to repeat for each iteration
- **Update**
  - Modify the variable(s) related to the condition
Type 1: while Loops

- A while loop is essentially a repeating 'if' statement

```
// initialization (e.g. i = 0)

while (condition1) {  // if condition1 is true
    // Body: if condition1 is true
    // go to top, eval cond1 again
    // only gets here when cond1 is false

    int i=0;
    while (i < 1000) {
        cout << i << endl;
        i++;
    }

    // following statements

    // following statements

    // following statements
```

While loop printing 0 to 999
while vs. do..while Loops

- while loops have two variations: while and do..while
  - while
    - Cond is evaluated first
    - Body only executed if condition is true (maybe 0 times)
  - do..while
    - Body is executed at least once
    - Cond is evaluated
    - Body is repeated if cond is true

```
// While:
while(condition)
{
    // code to be repeated
    // (should update condition)
}

// Do while:
do {
    // code to be repeated
    // (should update condition)
}while(condition);
```
while Loop

- One way to think of a while loop is as a repeating 'if' statement
- When you describe a problem/solution you use the words 'until some condition is true' that is the same as saying 'while some condition is not true'
  - "Until they guess correctly" is the same as "while they do NOT guess correctly"

// guessing game
bool guessedCorrect = false;
if( !guessedCorrect )
{
    guessedCorrect = guessAgain();
}
// want to repeat if cond. check again
if( !guessedCorrect )
{
    guessedCorrect = guessAgain();
} // want to repeat if cond. check again

An if-statement will only execute once

// guessing game
bool guessedCorrect = false;
while( !guessedCorrect )
{
    guessedCorrect = guessAgain();
}

A 'while' loop acts as a repeating 'if' statement
Using Flow Charts to Find Loops

Draw out a flow chart of the desired sequence and look for the repetitive sequence:

Here we check at the end to see if we should repeat...perfect for a do..while loop:

```plaintext
do {
    accept_guess
}
while ( ! correct )
```
Finding the ‘while’ Structure

Draw out a flow chart of the desired sequence and look for the repetitive sequence

Here we check at the end to see if we should repeat...perfect for a do..while loop

But a while loop checks at the beginning of the loop, so we must accept one guess before starting:

```c
accept_guess while( ! correct )
{ accept_guess }
```

accept_guess
while( ! correct )
{ accept_guess }
Type 2: 'for' Loop

- 'for' loop
  - performs initialization statement once
  - checks the condition each iteration before deciding to execute the body or end the loop
  - performs the update statement after each execution of the body

```
for( init; condition; update)
{
    // executed if condition is true
} // go to top, do update, eval cond. again
// following statements
// only gets here when cond. is false
```
for Loop

- **Initialization stmt executed first**
- **Cond is evaluated next**
- **Body only executed if cond. is true**
- **Update stmt executed**
- **Cond is re-evaluated and execution continues until it is false**
- **Multiple statements can be in the init and update statements**
  - Separate with commas

```cpp
for(init stmt; cond; update stmt) {
    // body of loop
}
```

// Outputs 0 1 2 3 4 (on separate lines)
for(i=0; i < 5; i++){
    cout << i << endl;
}

// Outputs 0 5 10 15 ... 95 (on sep. lines)
for(i=0; i < 20; i++){
    cout << 5*i << " is a multiple of 5";
    cout << endl;
}

// Same output as previous for loop
for(i=0; i < 100; i++){
    if(i % 5 == 0){
        cout << i << " is a multiple of 5";
        cout << endl;
    }
}

// compound init and update stmts.
for(i=0, j=0; i < 20; i++,j+=5){
    cout << j << " is a multiple of 5";
    cout << endl;
}
```
for vs. while Loop

- 'while' Rule of thumb: Use when exact number of iterations is unknown when loop is started (i.e. condition updating inside the loop body)

- 'for' Rule of thumb: Use when number of iterations is known when loop is started (independent of loop body)

- Both can be converted to the other...try it on the right

```cpp
int x;
cin >> x;
for(i=0; i < x; i++){
    cout << 5*i << " ";
}
cout << endl;
```

```cpp
bool guessedCorrect = false;
while( !guessedCorrect )
{
    guessedCorrect = guessAgain();
}
```

Notice we cannot predict how many times this will run.

Though we don't know x we can say the loop will run exactly x times.
LOOP IDIOMS & PRACTICE
Map Idiom

- **Name**: Map
- **Description**: Convert (map) each value in a collection to another value
- **Structure**: Use a loop to process a series of input values and convert to the desired output values
  - Usually with a n-to-n input-output relationship
- **Example(s)**:
  - See examples on the right

```plaintext
for (/* loop thru each input */)
{
    // Get next input, x
    // Produce next output, f(x)
}
```

**Structure**

**Output the first n odd integers**
- **Input**: 0, 1, 2, ..., n-1
- **Output**: 1, 3, 5, ..., 2(n-1)+1

**Given a threshold of 70, indicate if students have passed a quiz**
- **Input**: 78, 61, 85, 93, 54
- **Output**: T, F, T, T, F

**Take the absolute value of each input**
- **Input**: -18, -13, 36, 2, -21
- **Output**: 18, 13, 36, 2, 21
Reduce Idiom

- **Name:** Reduce / Combine / Aggregate
- **Description:** Combine/reduce all elements of a collection to a single value
- **Structure:** Use a "reduction" variable and a loop to process a series of input values, combining each of them to form a single (or constant number of) output value in the reduction variable
  - An n-to-1 input-output relationship
- **Example(s):**
  - See example on the right

```cpp
// Declare reduction variable, r
// Set r to identity value
for(/* loop thru each input */)
{
    // Get next input, x
    // Update r using x
}
```

**Average a series of 4 numbers**

Input: 2, 3, 1, 8
Average: 3.5

```cpp
double sum = 0;
double x;
for(int i=0; i < 4; i++)
{
    cin >> x;
    sum += x;
}
cout << sum / 4.0 << endl;
```
Selection Idiom

- **Name**: Selection
- **Description**: Select a subset (possibly one or none) of elements from a collection based on a particular property
- **Structure**: Loop through each element and check whether it meets the desired property. If so, perform a `map`, `reduce`, or other other update operation.
- **Example(s)**:
  - Count all positive integers inputs

```java
// declare/initilize any state variables
// needed to track the desired result

// loop through each instance
for( /* each input, i */ ) {
    // Check if input meets the property
    if(property is true for i) {
        // Update state (variables) as needed
    }
}
// Output the state variables
```

**Structure**

**Count Positive Integers**

**Input**: 2, -3, -1, 8
**Output**: 2
Exercises

• In-class exercises:
  – countodd
  – liebnizapprox
  – wallis
  – revdigits
Loop Practice

• Write a for loop to compute the first 10 terms of the Liebniz approximation of $\pi/4$:
  
  • $\pi/4 = 1/1 - 1/3 + 1/5 - 1/7 + 1/9 \ldots$
  
  • Tip: write a table of the loop counter variable vs. desired value and then derive the general formula

• In-class exercise:
  
  – liebnizapprox

<table>
<thead>
<tr>
<th>Counter (i)</th>
<th>Desired</th>
<th>Pattern</th>
<th>Counter (i)</th>
<th>Desired</th>
<th>Pattern</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>+1/1</td>
<td>for(i=0;i&lt;10;i++) Fraction:</td>
<td>1</td>
<td>+1/1</td>
<td>for(i=1; i&lt;=19; i+=2) Fraction:</td>
</tr>
<tr>
<td>1</td>
<td>-1/3</td>
<td></td>
<td>3</td>
<td>-1/3</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>+1/5</td>
<td>+/- =&gt;</td>
<td>5</td>
<td>+1/5</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td></td>
<td>...</td>
<td>...</td>
<td>+/- =&gt;</td>
</tr>
<tr>
<td>9</td>
<td>-1/19</td>
<td></td>
<td>19</td>
<td>-1/19</td>
<td></td>
</tr>
</tbody>
</table>
Loop Practice

• Write for loops to compute the first 10 terms of the following approximations:

  – $e^x$: $1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} \ldots$
    
    • Assume 1 is the 1\textsuperscript{st} term and assume functions
      – fact(int n) // returns n!
      – pow(double x, double n) // returns $x^n$

  – Wallis:
    
    • $\pi/2 = \frac{2}{1} \ast \frac{2}{3} \ast \frac{4}{3} \ast \frac{4}{5} \ast \frac{6}{5} \ast \frac{6}{7} \ast \frac{8}{7} \ldots$
    • In-class Exercise
      – wallisapprox
20-second Timeout: Chunking

- Right now you may feel overwhelmed with all the little details (all the parts of a for loop, where do you need semicolons, etc.)
- As you practice these concepts they will start to "chunk" together where you can just hear "for loop" and will immediately know the syntax and meaning
- Chunking occurs where something more abstract takes the place of many smaller pieces

"Chunking" allows complex tasks to become simple

Tooth Brushing
- Turn on faucet
- Rinse toothbrush
- Turn off faucet
- Remove cap from tooth paste
- Apply toothpaste to brush
- Replace cap
- Insert brush into mouth
- Brush teeth for 2 minutes
- Turn on faucet
- Spit
- Rinse mouth
- Rinse brush

https://designbyben.wordpress.com/tag/chunking/
On your own time, practice tracing the following loops

TRACING EXECUTION 1
Tracing Exercises (Individually)

- To understand a loop's execution make a table of relevant variable values and show their values at the time the condition is checked.
- If the condition is true perform the body code on your own (i.e. perform specified actions), do the update statement, & repeat.

```cpp
int i;
cout << "For 1: " << endl;
for(i=0; i < 5; i++){
    cout << i << " ";
}
cout << i+10 << endl;
```

<table>
<thead>
<tr>
<th>i (at condition check)</th>
<th>Actions of body</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>&quot;0 &quot;</td>
</tr>
<tr>
<td>1</td>
<td>&quot;1 &quot;</td>
</tr>
<tr>
<td>2</td>
<td>&quot;2 &quot;</td>
</tr>
<tr>
<td>3</td>
<td>&quot;3 &quot;</td>
</tr>
<tr>
<td>4</td>
<td>&quot;4 &quot;</td>
</tr>
<tr>
<td>5</td>
<td>-</td>
</tr>
<tr>
<td>Done</td>
<td>&quot;0 1 2 3 4 15\n&quot;</td>
</tr>
</tbody>
</table>
Tracing Exercises (for 2-4)

• Perform hand tracing on the following loops to find what will be printed:

```cpp
int i;
cout << "For 2: " << endl;
for(i=0; i < 5; i++){
    cout << 2*i+1 << " ";
}
cout << endl;
```

```cpp
int i, j=1;
cout << "For 3: " << endl;
for(i=0; i < 20; i+=j){
    cout << i << " ";
    j++;
}
cout << endl;
```

```cpp
int i, j=1;
cout << "For 4: " << endl;
for(i=0; i > 0; i--){
    cout << i+j << " ";
    i = i/2; j = j*2;
}
cout << endl;
```

Answers at end of slide packet
Tracing Exercises (for 5-6)

- Perform hand tracing on the following loops to find what will be printed:

```cpp
int i = 3;
char c = 'a';

cout << "For 5: " << endl;
for( ; c <= 'j'; c+=i ){
    cout << c << " " ;
}
cout << endl;
```

```cpp
double T = 8;

cout << "For 6: " << endl;
for(i=0; i <= T; i++){  // Force rounding to 3 decimal places
    cout << fixed << setprecision(3);
    // Now print the number
    cout << sin(2*M_PI*i/T) << endl;
}
```

Answers at end of slide packet
Tracing Exercises (while 1-2)

- Perform hand tracing on the following loops to find what will be printed:

```cpp
int i=15, j=4;
cout << "While loop 1: " << endl;
while( i > 5 && j >= 1){
    cout << i << " " << j << endl;
    i = i-j;
    j--;
}
```

```cpp
int i=1; j=1;
cout << "While loop 2: " << endl;
while( i || j ){
    if(i && j){
        j = !j;
    } else if( !j ){
        i = !i;
    }
    cout << i << " " << j << endl;
}
```

Answers at end of slide packet
Tracing Exercises (while 3)

- Perform hand tracing on the following loops to find what will be printed:

```cpp
bool found = false;
int x = 7;
while( !found ){
    if( (x%4 == 3) &&
        (x%3 == 2) &&
        (x%2 == 1) )
    {
        found = true;
    }
    else {
        x++;
    }
}
cout << "Found x = " << x << endl;
```

Answers at end of slide packet
break statement

• break
  – Ends the current loop [not if statement] immediately and continues execution after its last statement

• Consider two alternatives for stopping a loop if an invalid (negative) guess is entered

```cpp
bool done = false;
while ( done == false ) {
  cout << "Enter guess: " << endl;
  cin >> guess;
  if( guess < 0 )
    done = true;
}
else {
  // Process guess
}
```
continue statement

• continue
  – Ends the current loop [not if statement] immediately and continues execution after its last statement

• Consider two alternatives for repeating a loop to get a new guess if an invalid (negative) guess is entered
  – Often continue can be eliminated by changing the if condition

```cpp
bool done = false;
while ( done == false ) {
    cout << "Enter guess: " << endl;
    cin >> guess;
    if( guess >= 0 ) {
        // Process Guess
    }
}
```

```cpp
bool done = false;
while( done == false) {
    cout << "Enter guess: " << endl;
    cin >> guess;
    if(guess < 0){
        continue;
    }
    // Process guess (only here if guess>=0)
}
```
Single Statement Bodies

- An if, while, or for construct with a single statement body does not require { ... }
- Another if, while, or for counts as a single statement

```java
if (x == 5)
    y += 2;
else
    y -= 3;

for(i = 0; i < 5; i++)
    sum += i;

while(sum > 0)
    sum = sum/2;

for(i = 1 ; i <= 5; i++)
    if(i % 2 == 0)
        j++;
```
The Loops That Keep On Giving

• There's a problem with the loops below
• We all write "infinite" loops at one time or another
• Infinite loops never quit
• When you do write such a program, just type "Ctrl-C" at the terminal to halt the program

```cpp
#include <iostream>
using namespace std;
int main()
{
    int val;
    bool again = true;
    while(again = true){
        cout << "Enter an int or -1 to quit";
        cin >> val;
        if( val == -1 ) {
            again = false;
        }
    }
    return 0;
}
```

```cpp
#include <iostream>
using namespace std;
int main()
{
    int i=0;
    while( i < 10 ) {
        cout << i << endl;
        i + 1;
    }
    return 0;
}
```

http://blog.codinghorror.com/rubber-duck-problem-solving/
The Loops That Keep On Giving

• There's a problem with the loop below
• We all write "infinite" loops at one time or another
• Infinite loops never quit
• When you do write such a program, just type "Ctrl-C" at the terminal to halt the program

```cpp
#include <iostream>
using namespace std;
int main()
{
  int val;
  bool again = true;
  while(again == true){
    cout << "Enter an int or -1 to quit";
    cin >> val;
    if( val == -1 ) {
      again = false;
    }
  }
  return 0;
}
```

```cpp
#include <iostream>
using namespace std;
int main()
{
  int i=0;
  while( i < 10 ) {
    cout << i << endl;
    i = i + 1;
  }
  return 0;
}
```

http://blog.codinghorror.com/rubber-duck-problem-solving/
NESTED LOOPS
What Can Go Inside?

- What kind of code can we put in the body of a loop?
- ANYTHING...even other loops

```java
while (condition) {
    // What can go here?
}
```

```java
for( init; condition; update) {
    // What can go here?
}
```
Nested Loop Sequencing

- **Key Idea**: The inner loop runs in its entirety for each iteration of the outer loop.

```plaintext
while (cond1) {
    // code1
    while (cond2) {
        // code 2
    }
    // code3
}
```
Nested Loops Example 1

- When you write loops consider what the body of each loop means in an abstract sense
  - The body of the outer loop represents 1 game (and we repeat that over and over)
  - The body of the inner loop represents 1 turn (and we repeat turn after turn)

```cpp
int main()
{
    int secret, guess;
    char again = 'y';
    // outer loop
    while(again == 'y')
    {
        // Choose secret num. 0-19
        secret = rand() % 20;
        guess = -1;
        // inner loop
        while(guess != secret)
        {
            cout << "Enter guess: ";
            cin >> guess;
            cout << "Win!" << endl;
            cout << "Play again (y/n): ";
            cin >> again;
        }
        cout << "Play again (y/n): ";
        cin >> again;
    }
    return 0;
}
```
Nested Loops

- Inner loops execute fully (go through every iteration before the next iteration of the outer loop starts)

```cpp
#include <iostream>
using namespace std;

int main()
{
    for(int i=0; i < 2; i++){
        for(int j=0; j < 3; j++){
            cout << i << " " << j << endl;
        }
    }
    return 0;
}
```

Output:
Nested Loops

• Write a program using nested loops to print a multiplication table of 1..12

• Tip: Decide what abstract "thing" your iterating through and read the for loop as "for each thing" ...
  – For each row...
    • For each column...
      print the product

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>6</td>
<td>9</td>
</tr>
</tbody>
</table>

#include <iostream>

using namespace std;

int main()
{
    for(int r=1; r <= 12; r++){
        for(int c=1; c <= 12; c++){
            cout << r*c;
        }
    }
    return 0;
}

This code will print some not so nice output:
Nested Loops

- Tip: Decide what abstract "thing" your iterating through and read the for loop as "for each thing" ...
  - For each row ...
    - For each column...
      - print the product followed by a space
    - Print a newline

```cpp
#include <iostream>
using namespace std;

int main()
{
    for(int r=1; r <= 12; r++)
    {
        for(int c=1; c <= 12; c++)
        {
            cout << " " << r*c;
        }
        cout << endl;
    }
    return 0;
}
```

This code will still print some not so nice output:

```
1 2 3 4 5 6 7 8 9 10 11 12
2 4 6 8 10 12 14 16 18 20 22 24
```
Nested Loops

• Use the `setw` I/O manipulator to beautify the output

```cpp
#include <iostream>
#include <iomanip>
using namespace std;

int main()
{
    for(int r=1; r <= 12; r++){
        for(int c=1; c <= 12; c++){
            cout << setw(4) << r*c;
        }
        cout << endl;
    }
    return 0;
}
```
break and continue (Nested Loops)

- Break and continue apply only to the inner most loop (not all loops being nested)
  - Break ends the current (inner-most) loop immediately
  - Continue starts next iteration of inner-most loop immediately
- Consider problem of checking if a '!' exists anywhere in some lines of text
  - Use a while loop to iterate through each line
  - Use a for loop to iterate through each character on a particular line
  - Once we find first '!' we can stop

```cpp
bool flag = false;
while( more_lines == true ){
    // get line of text from user
    length = get_line_length(...);

    for(j=0; j < length; j++){
        if(text[j] == '!'){
            flag = true;
            break; // only quits the for loop
        }
    }
}
```

```cpp
bool flag = false;
while( more_lines == true && ! flag ){
    // get line of text from user
    length = get_line_length(...);

    for(j=0; j < length; j++){
        if(text[j] == '!'){
            flag = true;
            break; // only quits the for loop
        }
    }
}
```
Nested Loop Practice

• In class exercises: checkerboard and flag

• In class exercise: 5PerLineA
  – Try to print out the integers from 100 to 200, five per line, as in:
    100 101 102 103 104
    105 106 107 108 109
    ...  
    195 196 197 198 199
    200

• In class exercise: 5PerLineB and 5PerLineC each have an error. Wee what they print and determine the error.
MODULE 7: C LIBRARIES & RAND()
Preprocessor & Directives

• Somewhat unique to C/C++
• Compiler will scan through C code looking for directives (e.g. #include, #define, anything else that starts with '#' )
• Performs textual changes, substitutions, insertions, etc.
• #include <filename> or #include "filename"
  – Inserts the entire contents of "filename" into the given C text file
• #define find_pattern replace_pattern
  – Replaces any occurrence of find_pattern with replace_pattern
  – #define PI 3.14159
    Now in your code:
    \[ x = \text{PI}; \]
    is replaced by the preprocessor with
    \[ x = 3.14159; \]
#include Directive

- Common usage: To include “header files” that allow us to access functions defined in a separate file or library
- For pure C compilers, we include a C header file with its filename: `#include <stdlib.h>`
- For C++ compilers, we include a C header file without the .h extension and prepend a ‘c’: `#include <cstdlib>`

<table>
<thead>
<tr>
<th>C</th>
<th>Description</th>
<th>C++</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>stdio.h</td>
<td>C Input/Output/File access (printf, fopen, snprintf, etc.)</td>
<td>iostream</td>
<td>I/O and File streams (cin, cout, cerr)</td>
</tr>
<tr>
<td>cstdio</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>stdlib.h</td>
<td>rand(), Memory allocation, etc.</td>
<td>fstream</td>
<td>File I/O (ifstream, ofstream)</td>
</tr>
<tr>
<td>cstdlib</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>string.h</td>
<td>C-string library functions that operate on character arrays</td>
<td>string</td>
<td>C++ string class that defines the ‘string’ object</td>
</tr>
<tr>
<td>cstring</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>math.h</td>
<td>Math functions: sin(), pow(), etc.</td>
<td>vector</td>
<td>Array-like container class</td>
</tr>
<tr>
<td>cmath</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
rand() and RAND_MAX

• (Pseudo)random number generation in C is accomplished with the rand() function declared/prototyped in cstdlib
• rand() returns an integer between 0 and RAND_MAX
  – RAND_MAX is an integer constant defined in <cstdlib>
• How could you generate a flip of a coin [i.e. 0 or 1 w/ equal prob.]?
  
  ```c
  int r;
  r = rand();
  if(r < RAND_MAX/2){ cout << "Heads"; }
  ```

• How could you generate a decimal with uniform probability of being between [0,1]
  
  ```c
  double r;
  r = static_cast<double>(rand()) / RAND_MAX;
  ```
Seeding Random # Generator

• Re-running a program that calls rand() will generate the same sequence of random numbers (i.e. each run will be exactly the same)
• If we want each execution of the program to be different then we need to seed the RNG with a different value
• srand(int seed) is a function in <cstdlib> to seed the RNG with the value of seed
  – Unless seed changes from execution to execution, we’ll still have the same problem
• Solution: Seed it with the day and time [returned by the time() function defined in ctime]
  – srand( time(0) ); // only do this once at the start of the program
  – int r = rand();  // now call rand() as many times as you want
  – int r2 = rand(); // another random number
  – // sequence of random #'s will be different for each execution of program

Only call srand() ONCE at the start of the program, not each time you want to call rand()!!!

Approximate rand() function:
val = ((val * 1103515245) + 12345) % RAND_MAX;
Loop Practice

• Write a for loop to compute the first 10 terms of the Liebniz approximation of $\pi/4$:

  $\pi/4 = 1/1 - 1/3 + 1/5 - 1/7 + 1/9 \ldots$

• Tip: write a table of the loop counter variable vs. desired value and then derive the general formula

<table>
<thead>
<tr>
<th>Counter (i)</th>
<th>Desired</th>
<th>Pattern</th>
<th>Counter (i)</th>
<th>Desired</th>
<th>Pattern</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>+1/1</td>
<td>for(i=0; i &lt; 10; i++) Fraction: $1/(2*i+1)$</td>
<td>1</td>
<td>+1/1</td>
<td>for(i=1; i &lt;= 19; i+=2) Fraction: $1/i$</td>
</tr>
<tr>
<td>1</td>
<td>-1/3</td>
<td>$+/- \Rightarrow \text{pow}(-1,i)$ if(i is odd) neg.</td>
<td>3</td>
<td>-1/3</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>+1/5</td>
<td></td>
<td>5</td>
<td>+1/5</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td></td>
<td>...</td>
<td>...</td>
<td>$+/- \Rightarrow \text{if}(i%4==3)$ neg.</td>
</tr>
<tr>
<td>9</td>
<td>-1/19</td>
<td></td>
<td>19</td>
<td>-1/19</td>
<td></td>
</tr>
</tbody>
</table>

Table: Counter (i) vs. Desired Value

<table>
<thead>
<tr>
<th>Counter (i)</th>
<th>Desired</th>
<th>Pattern</th>
<th>Counter (i)</th>
<th>Desired</th>
<th>Pattern</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>+1/1</td>
<td>for(i=0; i &lt; 10; i++) Fraction: $1/(2*i+1)$</td>
<td>1</td>
<td>+1/1</td>
<td>for(i=1; i &lt;= 19; i+=2) Fraction: $1/i$</td>
</tr>
<tr>
<td>1</td>
<td>-1/3</td>
<td>$+/- \Rightarrow \text{pow}(-1,i)$ if(i is odd) neg.</td>
<td>3</td>
<td>-1/3</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>+1/5</td>
<td></td>
<td>5</td>
<td>+1/5</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td></td>
<td>...</td>
<td>...</td>
<td>$+/- \Rightarrow \text{if}(i%4==3)$ neg.</td>
</tr>
<tr>
<td>9</td>
<td>-1/19</td>
<td></td>
<td>19</td>
<td>-1/19</td>
<td></td>
</tr>
</tbody>
</table>
Tracing Answers

For 1:
0 1 2 3 4 15

For 2:
1 3 5 7 9

For 3:
0 2 5 9 14

For 4:
11 6 5

For 5:
a d g j

For 6:
0.000
0.707
1.000
0.707
0.000
-0.707
-1.000
-0.707
-0.000

While loop 1:
15 4
11 3
8 2
6 1

While loop 2:
1 0
0 0

While loop 3:
Found x = 11