Game Plan

- A very brief review of all of the topics covered in the class

- Examples to help demonstrate the use of these concepts

- Big examples similar to what to expect on the final (hopefully!)
Final Time and Format

- Wednesday, April 27. 2:00 – 5:00 pm.
- Departmental

- 45 Multiple choice questions (/45)
- Trace a Program (/15)
- 2 Large programming questions (/60 each)
Hello World

// This program prints “Hello World”.

#include <iostream>

using namespace std;

int main()
{
    cout << "Hello world!" << endl;
    return 0;
}
Includes

#include <iostream>
#include <fstream>
#include <string>
#include <iomanip>
#include <cmath>
#include <cstdlib>
#include <ctime>
#include <cassert>
#include <cctype>
#include <cstring>
#include <cassert>
#include <cctype>
Namespaces

With Namespaces

#include <iostream>

using namespace std;

int main()
{
    cout << "Hello world!" << endl;
    return 0;
}
Namespaces

Without Namespaces

#include <iostream>

int main()
{
    std::cout << "Hello world!" << std::endl;
    return 0;
}
Printing to the Screen

cout << "This is some text!" << endl;

Prints...

This is some text!
Comments

// This is a line comment.

/*
This is a block comment.
It can span more than one line!
*/
Primitive Variables

char
int
double
float
bool
void
Integer

A whole (counting) number.

Eg. ...-2, -1, 0, 1, 2 ...

Eg.
int someNumber = 7;
Other int types

- unsigned int – use the bit patterns that would represent negative integers to instead represent more positive integers
- short, long int – use fewer or more bits to represent the value
- Typical signed range (32 bit):
  -2 billion .... 2 billion
Char

Stores a single character
Really just a small integer cross referenced with the ASCII table
Char literals are surrounded by single quotes

Eg.
char someLetter = ‘a’;
Float, Double

 Represents a number with a decimal value. Not always precise*. Double is more precise than float ** Be careful with division involving floats and integers

Eg.

float someDecimal = 1.378;
What do these print out?

```cpp
int x = 10, y = 3;
float a = 10.0, b = 3.0;

cout << x / y << endl;
cout << x / b << endl;
cout << a / y << endl;
cout << a / b << endl;
```
What do these print out?

```cpp
int x = 10, y = 3;
float a = 10.0, b = 3.0;

cout << (int) x / (int) y << endl; // 3
cout << (int) x / (float) b << endl; // 3.33
cout << (float) a / (int) y << endl; // 3.33
cout << (float) a / (float) b << endl; // 3.33
```
Float Addition

```cpp
float f = 1.0 / 3.0;
cout << f;
float g = f + f + f;
cout << g;
```
Boolean

Can hold 2 values:
  true
  false
Created by relational operators.

Eg.
bool someFlag = true;
Conversion

Explicit conversion (casting) => Tell the compiler what to do
int x = 3;
float y = (float) x;                  // c style
float y = float (x)                   // function style
float y = static_cast<float> (x);    // c++ style

Implicit conversion (coercion) => Let compiler decide what to do.
int x = 3;
float y = x;
Casting rules

float -> int will truncate the bits after the decimal.

char -> int will return the ASCII value, and vice versa.

anything -> bool will return true if that value is non-zero, and false if that value is zero.
Casting Puzzle

```c
float f = 107.82;
char c;

c = (char) (int) f;
```
More casting puzzles

float y = 0.3210;
bool b;

b = (bool) (int) y;
One last casting puzzle

float y = 0.3210;
bool b;

b = (bool) y;
What type would you use to...

Signal if a program had started?
Count the number of letters in a word?
Store a middle initial?
Store the age of a person?
Store the mass a person?
Store the mass of a grain of sand?
Store a grade?
Make your own types (sorta)

typedef int myType
    -> myType is now an alias for int.

enum primaryColors
{   RED, BLUE, YELLOW   };
    -> primaryColors can hold only these three values.
    -> Really just labels for ints, but easy to read.
Constants

Constants are like variables but can’t be changed. Declared in header section.

Eg.
const int DAYS_IN_APRIL = 30;
int main ()
{
 ...

char functions and `<cctype>`

C++ provides the ability to perform manipulations on the char type, which are included in the cctype header

2 types of functions in `<cctype>`
- Conversion functions
- Classification functions
cctype – classification functions

- functions which take a char and return a boolean (true or false) if the character meets some criteria
- isalpha, isdigit, islower, ispunct, isspace, isupper
- eg.
  
  bool isADigit = isdigit ('3');
What do the following hold?

```cpp
bool isItDigit = isdigit ('a');
bool isItUpper = isupper ('Z');
bool isItLower = islower ('A');
bool isItSpace = isspace (' ');
bool isItPunct = ispunct ('!');
```
What do the following hold?

```cpp
bool isItDigit = isdigit ('a'); // false
bool isItUpper = isupper ('Z'); // true
bool isItLower = islower ('A'); // false
bool isItSpace = isspace (' '); // true
bool isItPunct = ispunct ('!'); // true
```
cctype conversion functions

- Functions which take a char and return a possibly converted char
- Performs conversion only if necessary

toupper, tolower eg. char upperA = toupper('a');
What do the following variables hold?

```c
char a = toupper ('A');
char b = toupper ('b');
char c = tolower ('C');
char d = tolower ('d');
char e = tolower ('6');
```
What do the following variables hold?

```c
char a = toupper (‘A’); // A
char b = toupper (‘b’); // B
char c = tolower (‘C’); // c
char d = tolower (‘d’); // d
char e = tolower (‘6’); // 6
```
Char Comparisons

Char comparisons are based on the values in the ASCII table:

Some important values

- ‘A’ = 65, ‘B’ = 66, … ‘Z’ = 90
- ‘a’ = 97, ‘b’ = 98, … ‘z’ = 122
- ‘0’ = 48, ‘1’ = 49, … ‘9’ = 57
- ‘ ‘ = 32
Char comparison implications

- Based on these values, we can make the following generalizations:
  - Uppercase letters < lowercase letters
  - Numbers < letters
  - If letters are the same case, a letter earlier in the alphabet < a later letter.
String

Not a primitive data type, a class
Must `#include <string>`
Represents an array of chars
Has many useful built in functions
Literals are enclosed in double quotes.

Eg.
```
string someLetters = "Whaaaaaat?!";
```
String Functions

Methods built into the string class
length / size
find
substr
at / array-style char access (myString[3])

Use dot notation, i.e.: myString.length();
String Examples

```cpp
string name = "Homer J Simpson";

char m = name.at(4)  // m = 'r'
int middle = name.find("J")  // middle = 6
int length = name.length()  // length = 15
string sub = name.substr(2, 5);
    // sub = "mer J"
```

How would you get the middle initial?
Middle initial...

```cpp
string name = "Homer J Simpson"

string middleInitial =
    name.at( name.find(' ') + 1 );
```
Concatenation

Append one string (or a string and a char) together. Returns a string.

Eg.

```java
string s = "Hello";
string t = "World";
String c = s + t;    // "HelloWorld"
```
String comparison

- When comparing strings, there are 4 rules that are followed:
- 1. Strings are the same length and contain the same characters – EQUAL
- 2. Strings are of the same length but contain different characters:
  - Perform a char comparison on the first character which is unequal and return the result.
String comparison

3. Strings are of different lengths, but contain the same characters in the overlap – The string with fewer characters is less than the string with more characters.

4. Strings are of different lengths but an inequality exists in the characters of the strings:
   - Perform a char comparison on the first character which is different and return the result.
Char Arrays

- We don’t need to use the `<string>` class to have arrays of characters.

- Could you build your own representation of this class? Could you implement some of these functions (substr, find, etc.)
Math Operators

+ Add
- Subtract
* Multiply
/ Divide (Integer or Floating Point)
% Modulus Division

(Get the remainder of two ints)
Math Examples

int x = 7, y = 3, z;

z = x + y;     // z = 10;
z = x – y;      // z = 4
z = x / y;     // z = 2
z = x % y;     // z = 1
z = x + y * y – x / y // z = 7 + 9 – 2 = 14
Unary Operators

!x           Not x
x++          Postfix Increment x
x--          Postfix Decrement x
++x          Prefix Increment x
--x          Postfix Decrement x
Prefix Vs Postfix

int x = 3;
int y;
y = x++;

At this point, what do x and y equal?
Prefix Vs Postfix

```java
int x = 3;
int y;
y = x++; // x = 4; // y = 3;
```

At this point, what do x and y equal?

```java
// x = 4;
// y = 3;
```
Many ways of doing the same thing

How many ways can you increment x?

x = x + 1;

x = 1 + x;

x += 1;

x++;  

++x;

Even though one way makes sense to you, it’s good to learn other ways to understand other people’s code. (Also helps on exams)
Relational Operators

- `a == b` is `a` equal to `b`?
- `a != b` is `a` not equal to `b`?
- `a > b` is `a` greater than `b`?
- `a < b` is `a` less than `b`?
- `a >= b` is `a` greater than or equal to `b`?
- `a <= b` is `a` less than or equal to `b`?

Will return true or false.
What would you use to…

Check if more than 6 people are in a room?
Check if there is exactly 31 days in a month?
Check if there are 3 or fewer days until your final?

There are many right answers: go with what works for you and explain it with comments.
Getting data from the user

1. Declare the variable to store the input
2. Get the input
3. Use the input

3 methods discussed
>> (stream extraction)
.get
getline
Steam Extraction Example

```cpp
#include <iostream>

using namespace std;

int main()
{
    int x;
    cout << "Please enter an integer: ";
    cin >> x;
    cout << "You entered " << x << "." << endl;
}
```
Get – get 1 character

#include <iostream>

using namespace std;

int main()
{
    char c;
    cout << "Please enter a character: ";
    cin.get(c);
    cout << "You entered " << c << "." << endl;
}
Getline – get a line

```cpp
#include <iostream>
#include <string>

using namespace std;

int main()
{
    string s;
    cout << "Please enter a sentence: ";
    getline(cin, s);
    cout << "You entered " << s << "." << endl;
}
```
# include <iostream>
# include <iomanip>

using namespace std;

int main()
{
    float y = 123.456;
    cout << y << endl; // 123.456
    cout << fixed << setprecision(1);
    cout << y << endl; // 123.5
}
More I/O Manipulations

fixed – keeps out of scientific notation
showpoint – always show decimal
setprecision(int) – only display the following number of digits (or, if fixed, decimal places)

setw(int) – make next print take up <int> spaces.
File I/O

Six Steps

> include #include <fstream>
> create file object: ifstream ifs;
> open file ifs.open("somefile.txt");
> check for errors if (!ifs) { return -1; }
> use the file like cin/cout ifs >> x;
> close the file ifs.close();
Boolean Operators

\[ x \&\& y \] Are \( x \) and \( y \) both true?

\[ x \mid\mid y \] Is \( x \) or \( y \) true?

\[ !y \] Is \( y \) NOT true (i.e. is \( y \) false)

\[ x \mid\mid !y \] Is \( x \) true or is \( y \) false?
Combinational Logic

... int day;
string month;
...
if (day == 29 && month == "feb") {
    cout << "Happy Leap Day" << endl;
} else if (month == "apr"
    || month == "may") {
    cout << "It’s Spring." << endl;
}
}
Short Circuit Evaluation

If two boolean expressions are combined with && (and), and the first one fails, the second will never evaluate.

If two boolean expressions are combined with || (or), and the first one succeeds, the second one will never evaluate.
Short Circuit Evaluation Eg.

```java
int x = 3;
int y = -3;
if (x < 0 && y < 0) { ... }
   // only x < 0 (false) needs to be evaluated.
if (x > 0 || y > 0) { ... }
   // only x > 0 (true) needs to be evaluated.

why do we care? We can make safeguards! Eg.
if ( x != 0 && y / x == 5) { ... }
```
Branching (if/then/else)

```cpp
int x;
cout << "Please enter a number: ";
cin >> x;
if (x > 10) {
    cout << "Your number is greater than 10." << endl;
} else if (x < 0) {
    cout << "Your number is negative." << endl;
} else {
    cout << "Your number is between 0 and 10." << endl;
}
```
Switch / Case

```cpp
char letter;
cin >> letter;
switch (letter) {
    case 'a':
        cout << "You entered 'a'" << endl;
        break;
    case 'b':
        break;
    case 'c':
        case 'd':
            cout << "You entered 'c' or 'd'." << endl;
            break;
    case default:
        cout << "You entered some other letter." << endl;
        break;
}
```
Loops!

For, While, Do-While.

For: All controls are in the first statement:

```cpp
for (int i = 0; i < 100; i++) {
    cout << "*";
}
```

How many stars will this print out?
Loooooooops!

While loop:
Enter into the loop while condition is true, exit while false. You are responsible for ensuring the exit is possible.

```cpp
int i = 10;
while (i < 30) {
    cout << i << endl;
    i++; // don’t miss this step!
}
```
Loooooooooooooooooops!

While loops can be controlled by any logic, not just counters...

```java
int lives = 3;
boolean gameOver = false;
while (!gameOver) {
    ...
    if (lives == 0) {
        gameOver = true;
    }
}
```
Loooooooooooooooooooops!

Typical exit conditions for loops are...

- 1. count controlled loops
- 2. sentinel loops
- 3. flag loops
- 4. file loops
Looolllllllllllllllllllllllllloooops

Do-While loops always run through the body once

```java
int x = 10
do {
    x++;
} while (x < 10);
// what’s x?
```

While loops can skip the body completely:

```java
int x = 10
while (x < 10) {
    x++;
}
// what’s x?
```
Advanced loop control

You can modify the flow of loops using the break and continue keywords.

break – prematurely exit the loop regardless of conditions
continue – skip the rest of the body and go right to the next iteration.
Abuse of these keywords creates Spaghetti Code!
Loop Tracing

Eg. What is the output of the following loop?

```cpp
int x = 0;
while (x < 7) {
    cout << "*";
    x ++;
}
```
More Loop Tracing

Eg. What is the output of the following loop?

```cpp
int x = 1;
while (x < 10) {
    cout << "*";
    x = x * 2;
}
```
Last Loop Tracing

Eg. What is the output of the following loop?

```cpp
int x = 0;
while (x < 10) {
    cout << "*";
    x = x * 2;
}
```
Weird loop termination

Eg. What is the output of the following loop?

```cpp
for (unsigned u = 10; u >= 0; u--) {
    cout << u << endl;
}
```
Weird loop termination

Eg. What is the output of the following loop?

```cpp
for (float f=0.333; f < 1; f+=0.333) {
    cout << f << endl;
}
```
Weird Loop Termination

Eg. What is the output of the following loop?

```cpp
for (float f=0.1; f < 1; f+=0.1) {
    cout << f << endl;
}
```
Nested Loops

Eg. What does the following print out

```cpp
for (int x = 0; x < 5; x++) {
    for (int y = 0; y < 20; y++) {
        cout << "*";
    }
    cout << endl;
}
```
More nested loops

// What does this print?
for (int i = 1; i < 20; i++) {
    for (int j = 1; j < 20; j++) {
        if (i == j) {
            cout << "*";
        } else {
            cout << " ";
        }
    }
    cout << endl;
}
More nested loops

// What does this print?
for (int i = 1; i < 20; i++) {
    for (int j = 1; j < 20; j++) {
        if (i >= j) {
            cout << "*";
        } else {
            cout << " ";
        }
    }
    cout << endl;
}
Functions

Basic structure is:

<return type> name ( <parameters> )
{
    // body of your function goes here

    return <some value>

}
Why use functions?

- Break down your code into manageable chunks.
- Easy to test (unit testing)
- Reusable
- Can specify how data goes in and out
- Get better grades
Void function

// print a star to the terminal
void printAStar()
{
    char star = ‘*’; // a local variable
    cout << star;
    return; // for void functions this is optional
}
Function scope

Your function only has access to variables you create within the function, those passed in or out, or “scary global variables”.

When your function starts, all its locals variables are created. When it exits, all its local variables are gone.
Static Function Variables

If you want a function to remember its value after it closes, use the static keyword
eg.

```java
int simpleCounter() {
    static int numberOfTimesCalled = 0;
    return numberOfTimesCalled ++;
}
```
Data in and out of functions

Functions are like sub programs working inside your program. They are isolated and only have access to data you give them.

1. Define if you need data to go in, out, in/out or neither.
2. Data that goes in is sent by parameters
3. Data that goes out is sent by return value or parameters if passed by reference.
In

```cpp
void printANumber (int inputValue) {
    cout << "The number is: "
        << inputValue << end;
    return;
}
```

Data comes into the function via parameter. Nothing comes back to the main. **printing to the console is not considered out**
```cpp
#include <cstdlib>

int rollADie ()
{
    return rand() % 6;
}

// data only comes out of the function.
// has limited utility so far
```
In/Out (Value / Return)

#include <cmath>

float pythagoras (float a, float b) {
    return sqrt(a * a + b * b);
}

// data enters via parameters, leaves via return statement.
In/Out (Reference)

```c
void doubleThisValue (int &x) {
    x = x * 2;    // x *= 2
}
```

Pass by reference means that changes that occur to the variable within the function will persist once the function terminates.
Pass by Reference vs. Value

```c++
void doubleIt (int &x) {
    x = x * 2;
}

int main () {
    int y = 4;
    doubleIt(y);
    cout << y; // what does y equal?
}
```
Pass by Value

```c
int returnItDoubled (int x) {
    return x * 2;
}
int main () {
    int y = 4;
    // how would you use this function to double y?
    ???
}
```
Pass by Value

```c
int returnItDoubled (int x) {
    return x * 2;
}

int main () {
    int y = 4;
    y = returnItDoubled(y);
}
```
Return Statements

A function must return the type that it is declared to return:

for void functions, this is nothing.

Once a function returns a value, it exits. You can put return values early to break out of the function quickly.
Early Return Function

float safeDiv (int numerator, int denominator)
{
    if (denominator == 0) {
        return -1.0; // some error value.
    }
    return (float) numerator / (float) denominator;
}
Prototyping

C++ needs to know the function name before it can call it. It must see that name before your main code. Solutions:

1. Write your function before your main (inline.) Works. Discouraged.
2. Prototype your functions before main.
   Eg. `void printAStar();`
   Encouraged.
Calling functions

Functions that have return values will be called from an assignment expression, eg.

```c
int returnsARandom();

int main() {
    int x = returnsARandom();
    ...
}
```
Calling Void Functions

Void functions (aka methods, procedures) get called from their own line.

Eg.

```c
void printAStar();
int main() {
    printAStar();
    ...
}
```
Calling functions with Parameters

Functions with parameters need to have one argument passed in for each parameter of that type

for pass by value, can be variables, literals, constants

for pass by reference, must be a variable for array parameters, must be an array of the correct dimensions
Good function commenting

// doubleIt
// purpose: returns the double of an integer value
// arguments: y - the value to be doubled
// returns: an int which is 2 * y
// precondition: y is an initialized int
// postcondition: the double of y is returned
int doubleIt (int y);
Arrays

Essentially just a bunch of variables of the same type all grouped together in your computer's memory.

// Create an Array of 30 ints.
int x [30];
Access individual ints using
x[0]  x[1]  x[2]  ...  x[28]  x[29]
Array bounds

Specified by using a literal or named constant. Named constant preferred.
Will not work with a variable.

Eg.
const int ARRAY_LENGTH = 30;
char y[ARRAY_LENGTH];
Why use Arrays?

You can access individual elements using square brackets
You can automatically access every index using loops
Eg. Say you wanted 9 integers with the following values…
0, 1, 2, 3, 4, 5, 6, 7, 8.
Without Arrays…

```java
int x0 = 0;
int x1 = 1;
int x2 = 2; ... Tedious tedious stuff.
```

Rule: when you find yourself repeating / copy and pasting a lot, there’s probably a smarter way to tackle this problem.
With Arrays

```c
int x[9]
for (int i = 0; i < 9; i++) {
    x[i] = i;
}
```

Much less tedious.
Array Declaration

```c
int x[] = { 10, 20, 30, 40, 50 };  
```
is equivalent to...
```c
int x[5];
x[0] = 10;
x[1] = 20;
x[2] = 30;
x[3] = 40;
x[4] = 50;
```
Other Array Considerations

You can make arrays of any type.
Arrays in functions are always pass by reference.
If you want an array that is protected from changes, use the “const” keyword.
C++ won’t warn you if you try to go out of bounds, but expect unpredictable results, annoying crashes.
Array Parameters in Functions

```c
void valueEqualsIndex (int arr[],
                       int size )
{
    for (int i = 0; i < size, i++) {
        array[i] = i;
    }
}
```
Multidimensional Arrays

To make arrays with additional dimensions, include additional square brackets, eg.

```c
int myMatrix[10][20];
```

this is an array with 10 rows, 20 columns.
Variable Scope and Lifetime

- **Scope** > all of the regions of code which can access a variable.

- **Lifetime** > the duration for which your variable lives.
Variable Scope and Lifetime

- The scope of an automatic variable is the block of code in which it was created, and all of the blocks of code nested in this function.
Variable Scope

... int x = 40;
if ( x > 30 ) {
    x = x + 1; // we can use x here.
}
...

...
Variable Scope in C++

- In C++, the lifetime of a variable is matched to the lifetime of the function (local) or file (global) in which it was declared.
- The only exception to this is function variables with the “static” keyword.
Ex. How long is x alive for?

```c
int main ( ) {
    myFunction ();
    // is x still alive here?
}
void myFunction () {
    int x = 0;
    cout << x << endl;
}
```
How to solve a programming problem on an exam

1. Don’t Panic.
2. Read the question.
3. Don’t panic.
4. Read the question again, and underline:
   - Requirements
   - Data types
   - Constraints
5. Further annotate the question if required
Problem solving strategy

6. Think of a high level algorithm for solving the problem in simple English

7. Write this algorithm down in the header comments of your program

8. Reread the question and check off that all of the requirements are met. If not, fix your algorithm now!
Problem solving strategy

9. On scrap paper, break down the algorithm into substeps, each sentence should target 1 – 3 lines of code. Identify potential functions.

10. At the beginning of the main function, declare any variables you might need. Leave room!

11. Use the algorithm on scrap paper as your comments to begin to write out the code.
Problem solving strategy

- 12. For each comment, write the corresponding C++ code until you have written a complete solution.
- 13. Reread the problem once more to see if you missed anything
- 14. Check over your code for common mistakes
Library of Common CS110 Errors

- Missing semicolons
- Incorrect loop logic
- Forgetting to open / check / close files
- Missing return statement
- Insufficient commenting
- Incorrect function parameter type
- Array out of bounds errors
- Int division instead of float division
Programming Questions

- The following slides contain some big programming questions.

- Try to apply the same problem solving approach which you would on the final.
1. Merge Arrays

Given the following arrays:

```c
int x[5] = {1, 2, 6, 8, 9};
int y[5] = {3, 4, 5, 7, 10};
int z[10] = {0};
```

Write a program which will populate z with the values from x and y in sorted order.
2. Char frequency Function

- Create a function with the following signature:
  
  ```c
  int countChars (string inputString, char inputChar);
  ```

- The function will return the number of times that character appears in the string.
3. Compression Preprocessor

- Huffman’s Code is a compression algorithm for text. You can think of it like a custom ASCII table which assigns a digit with a shorter binary representation to the characters which appear the most frequently in text.

- In order to use this algorithm, it is necessary to first count all of the characters in a text document.
3. Compression Preprocessor

- A file named “simpleText.txt” contains a series of letters, punctuation and spaces.
- We wish to count each of the alphabetical characters in this file.
- Create an array of 26 elements, with the index of each element representing the letter’s position in the alphabet. (ie. a = 0, b = 1, etc.)
- Count the number of times each letter appears in the file, and store it in the correct spot in the array.
3. Compression Preprocessor

Eg.

myArray[0] = 10; // this is the count for ‘a’
myArray[1] = 3; // this is the count for ‘b’
myArray[2] = 4; // this is the count for ‘c’
myArray[3] = 5; // this is the count for ‘d’
...

3. Compression Preprocessor

- Finally, print the contents of the array to the screen in the following format:

  \[
  \begin{align*}
  a &= 10 \\
  b &= 3 \\
  c &= 4
  \end{align*}
  \]

  Note: Ignore all characters which are not lower case ASCII characters.
4. Compute the average

Create a function which takes only one parameter, an array of floats of any size. Somewhere in the array will be the value 0. Compute the average of all the values before the first 0. Return the value as a double. Trust that a 0 will be encountered before the end of the array bounds.
5. Reverse in Place

Write a function that will reverse the order of an array in place, ie. Do not make a copy of the array. The prototype should be as follows:

```c
void reverseOrder (int arr[], int size);
```
6. Big database example

A text file named “minerals.dat” contains records for minerals in the following format:

<table>
<thead>
<tr>
<th>Name</th>
<th>Color</th>
<th>Mass (g)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Orthoclase</td>
<td>White</td>
<td>140.0</td>
</tr>
<tr>
<td>Plagioclase</td>
<td>Blue</td>
<td>60.3</td>
</tr>
<tr>
<td>Calcite</td>
<td>Green</td>
<td>30.8</td>
</tr>
</tbody>
</table>

Parse this data to create a new file named “averages.dat” which includes records for the minerals in the format

<table>
<thead>
<tr>
<th>Name</th>
<th>Average Mass (g)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Orthoclase</td>
<td>137.6</td>
</tr>
</tbody>
</table>

Assume no more than 10 unique minerals exist in the data
Strategy

Create parallel arrays of names, total sizes, total counts

> Grab the first mineral name, check if it exists in the array
> if so, increment the count and add to the total size
> if not, add it to the array.
> once complete, output the list to the output file.
7. Simple Sudoku Checker

- A file named “sudoku.txt” contains numbers in the following format:

  1 2 3 4 5 6 7 8 9
  6 7 8 9 1 2 3 4 5
  7 8 9 4 5 6 1 2 3

Write a program which will read in this file, and write to the screen which rows contain only unique numbers from 1 to 9.
8. Convert Between Hex and Binary

- Write a function that allows a user to enter a char which is a valid hex character (0 .. 9 and A .. F) and output the corresponding binary representation of that number to the screen.
- The letters can be upper or lower case
- If the character is invalid for hex, write an error statement.
## Decimal, Binary and Hexadecimal

<table>
<thead>
<tr>
<th>Decimal</th>
<th>Binary</th>
<th>Hexadecimal</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>10</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>11</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>100</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>101</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>110</td>
<td>6</td>
</tr>
<tr>
<td>7</td>
<td>111</td>
<td>7</td>
</tr>
<tr>
<td>8</td>
<td>1000</td>
<td>8</td>
</tr>
<tr>
<td>9</td>
<td>1001</td>
<td>9</td>
</tr>
<tr>
<td>10</td>
<td>1010</td>
<td>A</td>
</tr>
<tr>
<td>11</td>
<td>1011</td>
<td>B</td>
</tr>
<tr>
<td>12</td>
<td>1100</td>
<td>C</td>
</tr>
<tr>
<td>13</td>
<td>1101</td>
<td>D</td>
</tr>
<tr>
<td>14</td>
<td>1110</td>
<td>E</td>
</tr>
<tr>
<td>15</td>
<td>1111</td>
<td>F</td>
</tr>
<tr>
<td>16</td>
<td>10000</td>
<td>10</td>
</tr>
</tbody>
</table>
9. What’s in a name?

- Given the char array (char [20] rawData):
  
  ['p']['H']['i']['L']['i']['P'][' ']['i'][' ']['f']['R']['y'][' '][' ']
  ...

- Populate the following three arrays / variables as follows:

  - char firstName[20] <= ['P']['h']['i']['l']['i']['p'][' ']
  - middleInitial <= 'J';
  - char lastName[20] <= ['F']['r']['y'][' ']

- Capitalize the first letter and make the rest lower
Other Potential Questions:

- Something involving Math-geometry
- Definitely something with char arrays
- Something with nested loops
- Something like encryption or palindromes
- Pass by Reference
- Char comparison questions